(2)

FINAL TECHNICAL REPORT

FUZZIFICATION OF ASAT'S RULE BASED AIMPOINT SELECTION

by

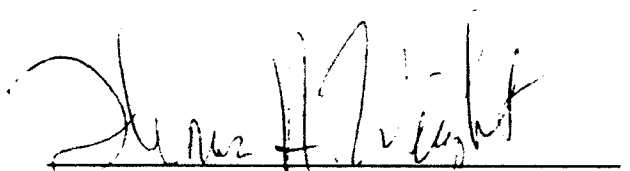
Thomas H. Weight, Ph.D.

Period Covered: 02 Nov 92 to 15 Jun 93

Contract DASG60-93-C-0005
for
U. S. Army Strategic Defense Command, Alabama 35807-3801

15 June 93

Dr. Weight and Associates
7005 E. Spring St.
Long Beach, CA 90808



Thomas H. Weight, Ph.D.
Principal Investigator
Dr. Weight and Associates

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY ---

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE ---

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release: SBIR report. distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
---

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Dr. Weight and Associates

6b. OFFICE SYMBOL (If applicatable)

7a. NAME OF MONITORING ORGANIZATION
U. S. Army Strategic Defense Command

6c. ADDRESS (City, State, and ZIP Code)
7005 E. Spring St.
Long Beach, CA 90808

7b. ADDRESS (City, State, and ZIP Code)
Contr & Acq Mgt Ofc., CSSD-CM-CB
P.O. Box 1500
Huntsville, AL 35807-3801

8a. NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL (If applicable)
---

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
DASG60-93-C-0005

8c. ADDRESS (City, State, and ZIP Code)
---

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
P665502	---	---	---

11. TITLE (Include Security Classification)
Fuzzification of ASAT's Rule Based Aimpoint Selection

12. PERSONAL AUTHOR(S)
Thomas H. Weight, Ph.D.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year,Month,Day)	15. PAGE COUNT
Final Technical	FROM 2 Nov 92 TO 15 June 93	1993, June, 15	66

16. SUPPLEMENTARY NOTATION
---

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify block numbers)
FIELD	GROUP	SUB-GROUP	fuzzy logic, fuzzy sets, fuzzy systems, aimpoint algorithms, kinetic energy weapon, ASAT, Anti-Satellite KEW
---	---	---	
---	---	---	

19. ABSTRACT (Continue on reverse if necessary and identify block number)

The aimpoint algorithms being developed at Dr. Weight and Associates are based on the concept of fuzzy logic. This approach does not require a particular type of sensor data or algorithm type, but allows the user to develop a fuzzy logic algorithm based on existing aimpoint algorithms and models. This provides an opportunity for the user to upgrade an existing system design to achieve higher performance at minimal cost.

Many projects have aimpoint algorithms which are based on "crisp" logic rule based algorithms. These algorithms are sensitive to glint, corner reflectors, or intermittent thruster firings, and to uncertainties in the a priori estimates of angle of attack. If these projects are continued through to a demonstration involving a launch to hit a target, it is quite possible that the crisp logic approaches will need to be upgraded to handle these important error sources.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Robert G. McIntosh

22b. TELEPHONE (Include Area Code)
205-955-4027

22c. OFFICE SYMBOL
CSSD-CM-CB

DD Form 1473, JUN 86 Previous editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

intentionally left blank

SUMMARY

The aimpoint algorithms being developed at Dr. Weight and Associates are based on the concept of fuzzy logic (or fuzzy sets). This approach does not require a particular type of sensor data or algorithm type, but allows the user to develop a fuzzy logic algorithm based on existing aimpoint algorithms and models. This provides an opportunity for the user to upgrade an existing system design to achieve higher performance at minimal cost.

Many projects have aimpoint algorithms which are based on "crisp" logic rule based algorithms. These algorithms are sensitive to glint, corner reflectors, or intermittent thruster firings, and to uncertainties in the *a priori* estimates of angle of attack. If these projects are continued through to a demonstration involving a launch to hit a target, it is quite possible that the crisp logic approaches will need to be upgraded to handle these important error sources.

This study looked at applying fuzzy logic to the ASAT aimpoint algorithm. The current baseline ASAT aimpoint algorithm is based on a "crisp" logic rule based algorithm. The ASAT project is currently considered to be at risk because the current baseline algorithms ignore glint. This study investigated retro-fitting the current baseline algorithms with fuzzy logic. The advantages of using fuzzy logic according to our study appear to be:

1. Broadly Useful: Fuzzy logic can be effectively applied to a broad spectrum of different type of sensors, algorithms, and problems.

2. Speed: This approach to using fuzzy logic would push all of the computationally intensive algorithms into the non-real time code. The result is that the real time code is extremely fast.

3. Timeliness: This approach can be developed into a useful working tool fairly quickly.

4. Robustness: Fuzzy logic is model free. As a consequence, fuzzy logic is insensitive to design errors and errors or uncertainty in the data being processed.

5. Accuracy: This study found that a factor of ten reduction in maximum aimpoint error resulted from the use of fuzzy logic.

Currently, Dr. Weight and Associates is involved in a continuing effort to produce a commercially marketable automated fuzzy logic development system which is capable of supporting the development of aimpoint algorithms for the ASAT, GBI, ERINT, E^{2}I, DEW, LADAR, THAADS, Brilliant Pebbles, Brilliant Eyes, etc..

PREFACE

This report describes the fuzzy logic aimpoint feasibility study and the software development undertaken at Dr. Weight and Associates under the Phase I SBIR contract DASG60-93-C-0005 sponsored by the U.S. Army Strategic Defense Command.

A Critical need currently exists for efficient aimpoint algorithms for kinetic energy weapons and for an effective automated tool to support the development of aimpoint rules during high stress, time constrained combat situations. In addition to aimpoints, there are problems in RADAR multipath, robotic control systems, sensor processing, and sensor fusion which can benefit directly from the use of fuzzy logic. In order for the benefits of this technology to be fully realized, the development of fuzzy logic will have to be automated in the relatively near future.

Our study of fuzzy logic aimpoint algorithms required the development of a generic data base of targets, and modelling the selection of aimpoints using several approaches. The two baseline ASAT aimpoint algorithms were compared to a human operator and to fuzzy logic. As a result of the preliminary comparison, one of the ASAT algorithms and the operator selection were rejected.

The fuzzy logic approach involves the use of existing crisp logic rule based algorithms. These rule based algorithms are applied to various scenarios involving factors such as glint and other sources of error. Fuzzy logic is used to determine which and to what extent each of these algorithms is to be applied. Fuzzy logic ends up taking the centroid (weighted average) of the aimpoints produced by the various rules. This centroid is seen to be much more accurate on average then any of the individual algorithms.

One of our findings is that there is a critical need for an automated approach to generating fuzzy systems. Current fuzzy development systems are basically operator intensive tools somewhat like a word processor: they are extremely useful but they do not design the system for you. We are proposing to continue work on an automated fuzzy development tool which will effectively support current and future military and commercial requirements in applications involving time critical, high pressure or combat situations.

We have not been required to work in a vacuum on this project. In addition to the several articles on the application of fuzzy logic in general and on fuzzy control systems in particular, there is much interest in both the public and private sectors. In particular, there is a lot of interest in the application of fuzzy logic to aimpoint algorithms. We would like to thank the managers and engineers of Lockheed, Hughes, McDonald Douglas, TRW, and especially Rockwell International for their time and their valuable suggestions.

TABLE OF CONTENTS

1.0 INTRODUCTION 1

2.0 FUZZY AIMPOINT ALGORITHM STUDY 4
2.1 Generic Targets 4
2.1.1 Three target types 4
2.1.2 Trajectory files 4
2.1.3 Software Tools 8
2.2 Image Intensity Maps 8
2.2.1 Point Spread Functions 8
2.2.2 Generating the images 12
2.2.3 Software tools 12
2.3 Glint 12
2.3.1 Glint ranges 12
2.3.2 Glint off different surfaces 12
2.4 Automatic Fuzzy System Generation 12
2.4.1 Adaptive Fuzzy Systems 14
2.4.2 Neural Networks 14
2.4.3 Basis Functions 15
2.4.4 Clustering Algorithms 15
2.4.5 Computed Neural Networks 15
2.5 Manual Rule Generation 15
2.6 Manual Aimpoint Generation 19
2.7 Automatic Scoring 19

3.0 STATUS OF ACCOMPLISHMENTS 20
3.1 Develop target profiles and intensity maps 20
3.2 Modify existing fuzzy logic software tool 20
3.3 Develop display based scoring system 20
3.4 Collect and analyze performance data 21
3.5 Future research and development 21

4.0 CONCLUSIONS AND RECOMMENDATIONS 25
4.1 Conclusions 25
4.1.1 Fuzzy logic will reduce maximum error 25
4.1.2 Fuzzy logic is robust 25
4.1.3 Fuzzy logic can be fast 25
4.1.4 Timely development of Fuzzy logic 25
4.1.5 Automating Fuzzy System Development 25
4.1.6 Underlying algorithm needs to be continuous . 25
4.1.7 Target needs to be blurred 26
4.2 Recommendations 26
4.2.1 Need automated development system 26
4.2.2 Errors in angle of attack 26
4.2.3 Computing neural networks 26
4.2.4 Investigate hybrid systems 27

5.0 POTENTIAL FOR COMMERCIALIZATION 28
5.1 Selling aimpoint algorithms 28
5.2 Selling automated development tools 28
5.3 Selling fuzzy logic and neural network expertise . . 28

Appendix A - Software listings . . . . . A - 1
filename: A_RULES . . . . . A - 2
filename: C_FUZZY . . . . . A - 5
filename: DISP . . . . . A - 10
filename: MK_ASAT . . . . . A - 13
filename: MK_FRN . . . . . A - 16
filename: MK_MAP . . . . . A - 18
filename: P_AIMPTS . . . . . A - 24
filename: SHOW . . . . . A - 26
filename: SHOW_3 . . . . . A - 29

LIST OF FIGURES

Figure 1: The three targets considered 5

Figure 2: The components of target one 6

Figure 3: A typical trajectory file 7

Figure 4: Computing ASAT aimpoint errors 9

Figure 5: Creating fuzzy aimpoints 10

Figure 6: We considered several point spread functions 11

Figure 7: Justification for the range of glints used 13

Figure 8: Three sets of membership functions were computed . . 16

Figure 9: The membership functions determined how to weight the three versions of the ASAT rules 17

Figure 10: Errors resulting from three models 18

Figure 11: Close range relative errors 22

Figure 12: Middle range relative errors 23

Figure 13: Far range relative errors 24

1.0 INTRODUCTION

The aimpoint algorithms being developed at Dr. Weight and Associates are based on the concept of fuzzy logic. This approach builds on and essentially replaces "crisp" rule based algorithms with fuzzy rule based systems. This approach allows existing projects to retrofit their current baseline aimpoint algorithms with fuzzy algorithms at very little cost and risk.

The approach investigated uses existing algorithms which are applied to a broad spectrum of image models. These image models correspond to the various scenarios likely to be encountered on a mission e.g. low, medium, or high glint off of various solar panels. During an actual mission, image parameters are evaluated to determine to what extent each scenario applies. The aimpoints corresponding to each scenario are weighted and averaged to produce a final aimpoint.

Some of the advantages of this approach are robustness, low risk, and low cost:

a. Fuzzy logic is insensitive to many types of error including modeling, and design errors. In fact fuzzy logic has the important property that it is tolerant of substantial deviations from the optimal membership functions. The difference in the quality of the aimpoints produced using optimal or suboptimal membership functions are not substantially different. When we compared the results using the optimal membership functions and membership functions which are easily machine generated, the difference was found to be small.

b. The approach to using fuzzy logic which was investigated involves using the current baseline ASAT aimpoint algorithm and applying fuzzy logic to it. This approach has the advantage that the quality of the aimpoints produced is at least as good as the baseline algorithms. This reduces the design risk to where we have essentially nothing to lose and everything to gain by using fuzzy logic.

c. The fuzzy logic algorithms involve very little new code and very little in terms of processor capabilities: additional memory and throughput required are minimal. The additional requirements are well within what are considered reasonable design margins by modern systems engineering standards.

Some of the disadvantages of this approach are the need for an automated tool for the development of fuzzy systems and the relatively advanced stage of development of many projects.

a. An automated tool for fuzzy system design is going to be somewhat more sophisticated than the tool currently envisioned to develop the ASAT aimpoint algorithms. The current baseline algorithms tend to ignore issues like glint and errors in angle of attack. As a result, these algorithms are relatively simple and the procedure for developing the aimpoint rules is also relatively simple. The fuzzy logic approach does not ignore these sources of error, and the tool for developing the aimpoint rules will be that much more complicated.

b. The ASAT project is nearing that point at which it will be locked into the current algorithm set. In order to make substantial modifications to the baseline real-time algorithms, these modifications will need to be coded and tested within less than one year. This will require the development of the real-time fuzzy logic algorithms before that development tool is designed and tested.

The basic goal of our Phase I research was to investigate the use of fuzzy logic in the ASAT aimpoint algorithm. Section 2.0 describes, in some detail, the research and development efforts expended by Dr. Weight and Associates to meet the Phase I technical objectives and summarizes the important Phase I results.

Section 3.0 describes the objectives of the Pnase I contract. Under the current contract, the following objectives were completed at Dr. Weight and Associates:

a. Developed a data base of unclassified generic target profiles and intensity maps. We selected the most challenging target profile and used that for all of the test cases. After some experimentation, we selected a non standard point spread function. This has the advantage of avoiding any possible issues with security. The target profiles where converted into high resolution images corresponding to targets at various ranges and various aspect angles. These images were point spread and converted into the low resolution images used for analysis.

b. Modified existing fuzzy logic software tool. This modification consisted of the inclusion of the ASAT baseline algorithms. We experimented with both of the baseline algorithms (extent and centroid) and settled on the centroid algorithm for our study. This tool ultimately produced several variations of the baseline ASAT algorithms, with several point spread functions, and with several fuzzy logic membership functions.

c. Developed display based scoring system. Software was

developed to display both the target profiles and the images. The profiles and target trajectories were verified using one of these tools. The images where displayed in order to allow an operator to identify the aimpoint. This approach was found to be hopeless and was abandoned in favor of using the resources to investigate automating fuzzy system development.

d. Collected and evaluated simulation data. Most of the data collected involved holding all put one parameter constant. This was necessary because of the huge amount of image processing necessary to build the final images at ten different ranges. The results were plotted and compared along with the membership functions, and the Root Mean Square errors.

e. Investigated automating fuzzy system generation. Current research publications were reviewed to determine the various approaches. These approaches were analyzed to determine their advantages and disadvantages. The results of this effort were combined with the findings of our research, and an approach was developed which seems optimal for our customers perceived needs.

Section 4.0 gives a statement of conclusions and recommendations. We concluded that all project technical objectives were attained. This project shows that fuzzy logic is a powerful technique when applied to the problem of selecting aimpoints. It allows an evolutionary approach to algorithm development which capitalizes on the currently popular aimpoint algorithms. After reasonable algorithms are developed, fuzzy logic is then applied to provide the features of robustness, high reliability, and high accuracy. We have found that at extreme range the performance of the fuzzy logic algorithms degrade to the performance of the crisp logic algorithms, but at short range the fuzzy logic algorithms are far superior.

2.0 FUZZY AIMPOINT ALGORITHM STUDY

Dr. Weight and Associates approach to using fuzzy logic in the ASAT aimpoint algorithm has the capability of supporting a wide range of commercial and military projects. By allowing the customer to use their baseline crisp logic algorithms, this approach allows the application of fuzzy logic to projects with a minimum of risk and cost.

Our goal was to determine whether fuzzy logic could make a substantial improvement in the performance of the ASAT aimpoint algorithm in the presence of glint. In order to achieve this goal, we started with a fairly accurate model of the current ASAT baseline algorithms and images. These allowed us to measure the performance of the fuzzy logic algorithm relative to the ASAT baseline. After showing that fuzzy logic could improve aimpoint accuracy, it was then necessary to determine whether fuzzy logic would be practical from the stand point of being automatically developed.

2.1 Generic Targets.

It was necessary to provide a database of targets which represented a realistic cross section of reality. At the same time, it was required to avoid any information which could be construed as being classified. Finally, we settled on one target type which had the advantage of being the most error prone.

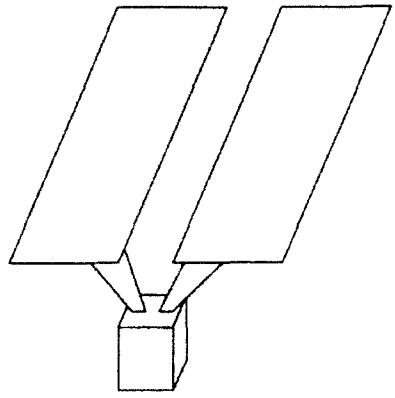
2.1.1 Three target types.

We considered three different target (see Figure 1) profiles with different bodies, reflective surfaces, and supporting members. After a little analysis it was determined that one profile offered the advantage of being most sensitive to glint. This target profile was used in all subsequent modeling and analysis. The target selected (target 1) has a relatively small body combined with two long solar panels (see Figure 2).

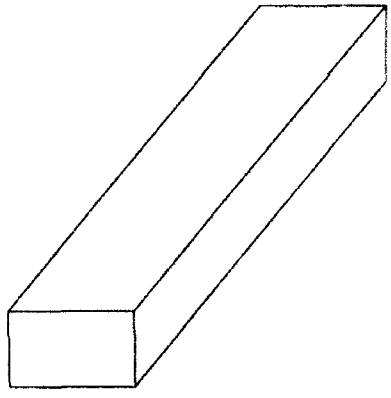
Each target configuration was described in terms of rectangular panels. Each target was allowed up to twenty different panels, and each panel could be oriented in any direction. If part of any panel is masked by some other panel then the masked part of the panel contributes no energy to the image.

2.1.2 Trajectory files.

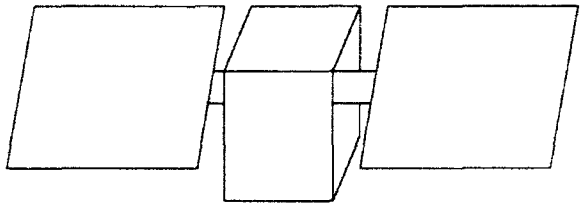
Trajectory files were defined which describe the target ranges, angles of attack, and glint. The trajectory file defines how many images are to be formed, and each image corresponds to a different range (see Figure 3). An initial setting for the angle of attack can be specified. Then for each image both the angle of attack and the glint can be changed. The angle of attack is changed by



Target #1



Target #2



Target #3

Figure 1: The three target profiles considered. Target number one was used because it had the greatest errors resulting from glint.

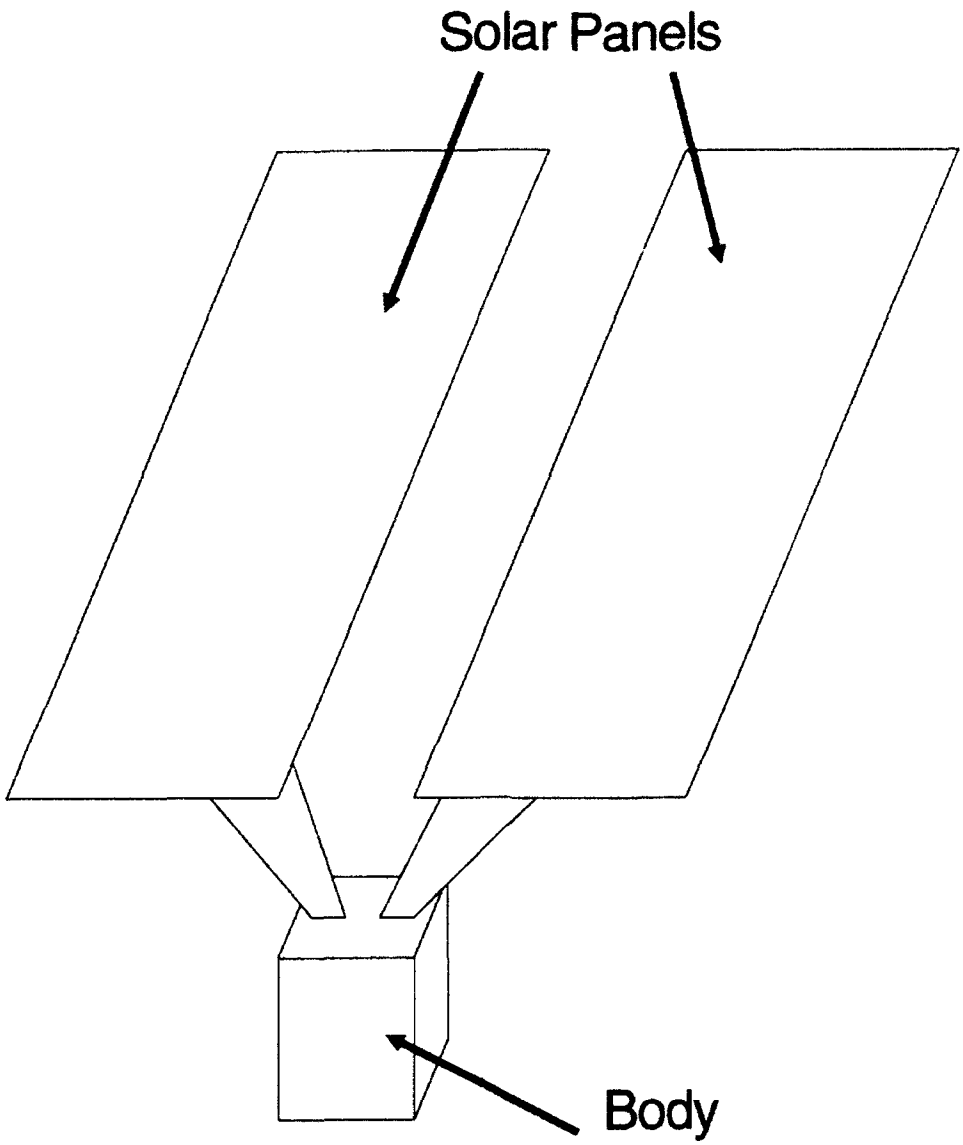


Figure 2: The components of the target. We assumed that the solar panels extending out the back of the target would be highly spectural surfaces which could be big glinting surfaces.

```
filename TGT06321.TDF
5 number of surfaces

         0.500000  0.500000 -0.500000
        -0.500000  0.500000 -0.500000
        -0.500000  0.500000  0.500000
         0.500000  0.500000  0.500000

         0.500000  0.500000 -0.500000
        -0.500000  0.500000 -0.500000
        -0.500000  0.500000  0.500000
         0.500000  0.500000  0.500000

         0.500000  0.500000 -0.500000
        -0.500000  0.500000 -0.500000
        -0.500000  0.500000  0.500000
         0.500000  0.500000  0.500000

         0.500000  0.500000 -0.500000
        -0.500000  0.500000 -0.500000
        -0.500000  0.500000  0.500000
         0.500000  0.500000  0.500000

         0.500000  0.500000 -0.500000
        -0.500000  0.500000 -0.500000
        -0.500000  0.500000  0.500000
         0.500000  0.500000  0.500000

0.00 0.00 0.00 aimpoint

10.0 -12.0  00.0 xrot yrot zrot

1 number of frames

1 number of images

1   0.000000 0 1 image rotation axes fcount

1 1  surface color
2 1  surface color
3 1  surface color
4 1  surface color
5 1  surface color
```

Rectangular surfaces are defined which together define the surface of the target. Each rectangular surface is defined by four points in three dimensional space.

In addition to these target surfaces, the aimpoint is also defined. This aim point allows later software to determine the errors computed by the various aim point algorithms.

The three values (xrot,yrot,zrot) define the init. orientation of the target: x,y,z are yaw, pitch, and roll respectively.

The number of frames is the number of pixel maps which will produced. The number of images is the number of different intensity and orientations which will define the trajectory.

Image, rotation,axes, and fcount give the image number, the angle of rotation, the axes of rotation and the number of maps to be from this image.

Finally, surface and color give surface number and its intensity.

Figure 3: A Typical Target Trajectory File. This file describes the target surfaces, and their brightness. It also describes the number of frames of image data to be produced and the sequence of rotations to be applied to the target from frame to frame.

specifying deltas from the previous value, while the required glint is simply specified. Glint is modeled by assigning an energy value to each target panel. Each panel can be assigned a value between one and fifteen independent of all other panels.

These trajectory files allow us to model targets through a series of images. The first image may correspond to an extreme long range. This long range can correspond to a target which covers a relatively small fraction of a pixel before applying the point spread function. A series of images are produced which represent the target from acquisition to just before Kill Enhancement Device (KED) deployment. At deployment of the KED, the target is an array of approximately five by five pixels.

2.1.3 Software Tools (see Figures 4 and 5).

Several software tools were developed to help debug the trajectory files. The first tool merely displays three views of the target: front, top, and side. This makes it fairly easy to verify the target file description of the target configuration. In addition, this tool displays each rectangular panel in the target in a different color independent of glint. This makes it easy to determine which surfaces are covered.

Another software tool displays the entire trajectory without actually building the image file. This tool allows the operator to review the trajectory and verify the image is correct in terms of angle of attack, and glint. This tool was necessary because actually building the image file was extremely time consuming - approximately eight minutes per image.

2.2 Image Intensity Maps.

There are several factors which influence the images generated. Because of limitations in time and resources, we decided to ignore several important parameters. In particular, parameters which are under the control of design engineers were for the most part ignored e.g. the point spread function. In addition, given that the signal to noise ratio gets extremely large closer to the target, we ignore noise.

2.2.1 Point Spread Functions (see Figure 6).

Although we eventually focused on glint, we initially started an investigation of point spread functions (PSF). This initial study included a wide range of functions which involved very little to a lot of blurring. It rapidly became clear that we could not include PSF's in our parametric study. We quickly settled on a nonstandard PSF which combined the two benefits of not being classified and of providing sufficient blurring to allow sub-pixel resolution.

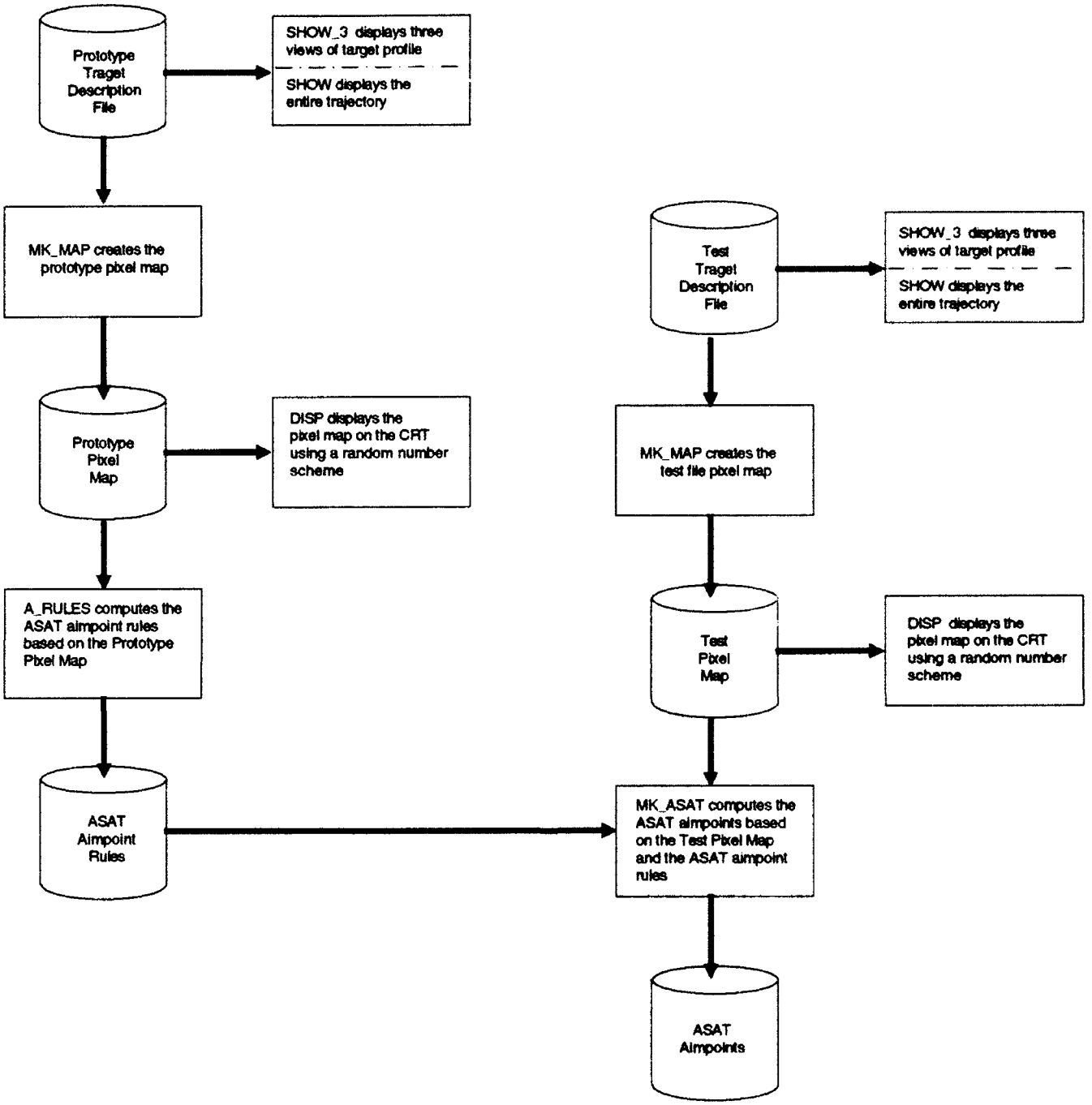


Figure 4: Computing ASAT aimpoint errors. These files and programs are used to compute the ASAT aim point rules, and the resulting ASAT aim point errors.

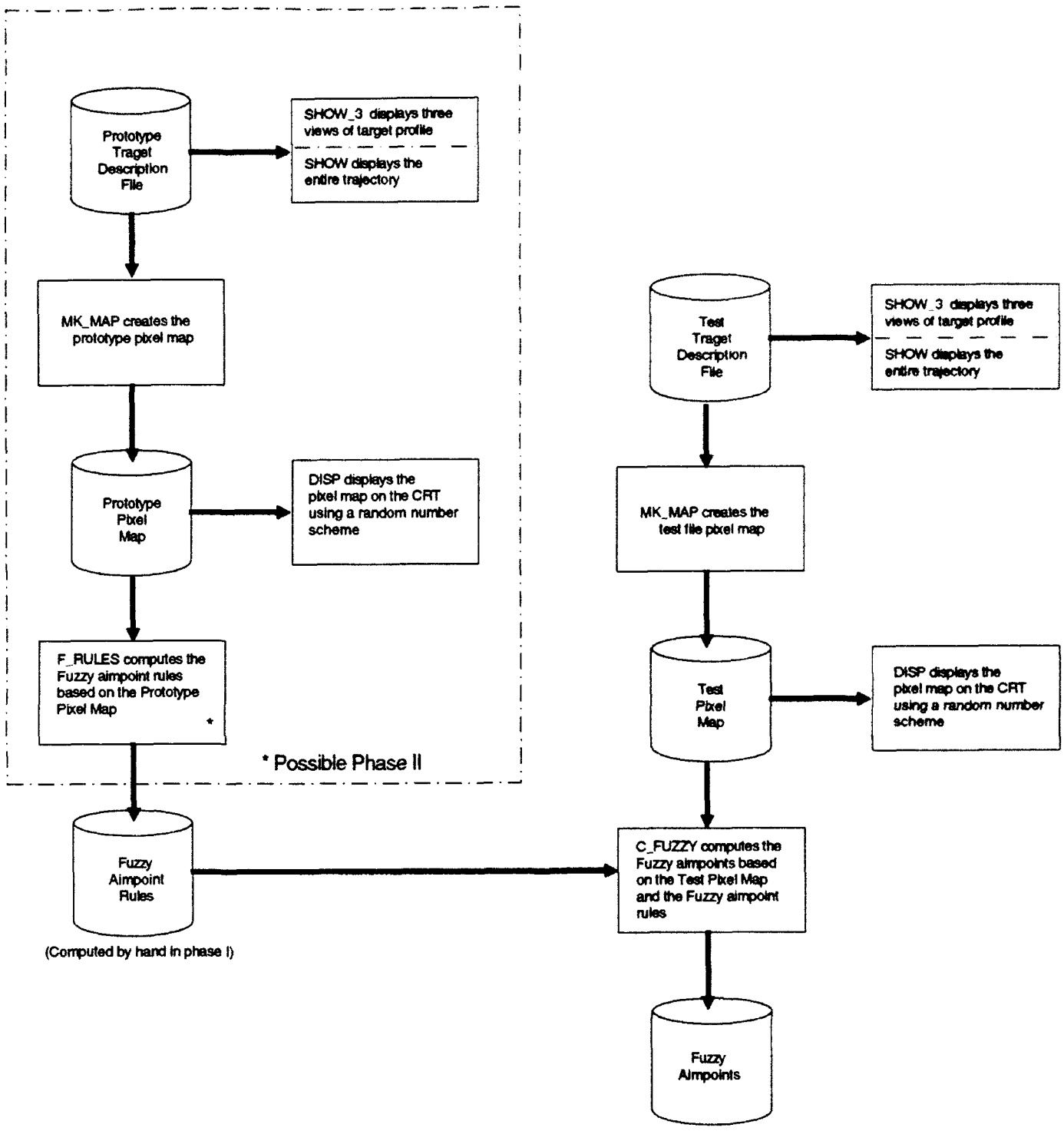


Figure 5: Creating fuzzy aimpoints. These files and programs are used to generate the fuzzy rules, and to compute the fuzzy aimpoints.

0.036	0.120	0.036
0.120	0.378	0.120
0.036	0.120	0. 036

Figure 6: We considered several point spread functions, but we finally selected the one shown above. This point spread function does not coincide with the ASAT PSF.

2.2.2 Generating the images.

The images were generated using a commercial graphics package. Initially, the images were formed on the computer screen. The different levels of glint were represented by false colors. After the image was formed then the pixels were allocated to various sub-pixel bins. The "energy" in each bin was accumulated, and blurred as defined by the programmable PSF. Finally, the total energy in each pixel was accumulated for output to an image file as a function of range.

2.2.3 Software tools (see Figures 4 and 5).

The tool which was developed accepts target description files and produces images files. We used a brute force approach which required very little programming effort, but which runs fairly slow on our computer. It is totally automated so that we were able to generate target image files during twenty hours runs while working on other tasks.

2.3 Glint.

Glint was specified as a major concern in the solicitation topic. As such, we addressed this issue first and in the most depth. Images were collected and analyzed corresponding to a complete range of glint coming of the "solar" panels.

2.3.1 Glint ranges (see Figure 7).

The target description file allows a dynamic range in the glint of fifteen. While actual glint may far exceed this dynamic range, it is adequate for our study. Graphs of the error generated by glint shows that nearly the maximum amount of error is achieved at approximately a factor of ten increase in the energy coming off the solar panel as compared to the target body.

2.3.2 Glint off different surfaces.

In our study, we focused on glint coming off both of the solar panels simultaneously. This is as opposed to glint off of one panel or the other. The justification foe ignoring these other combinations rests on the fact that at long range both panels appear to be collocated. At short range, one panel has essentially the same effect as an error in the angle of attack.

2.4 Automatic Fuzzy System Generation.

Most approaches to automatically generating fuzzy systems have problems which are related to training or to requiring an operator to predefine an initial system configuration. Although several learning paradigms have been identified, almost all of the

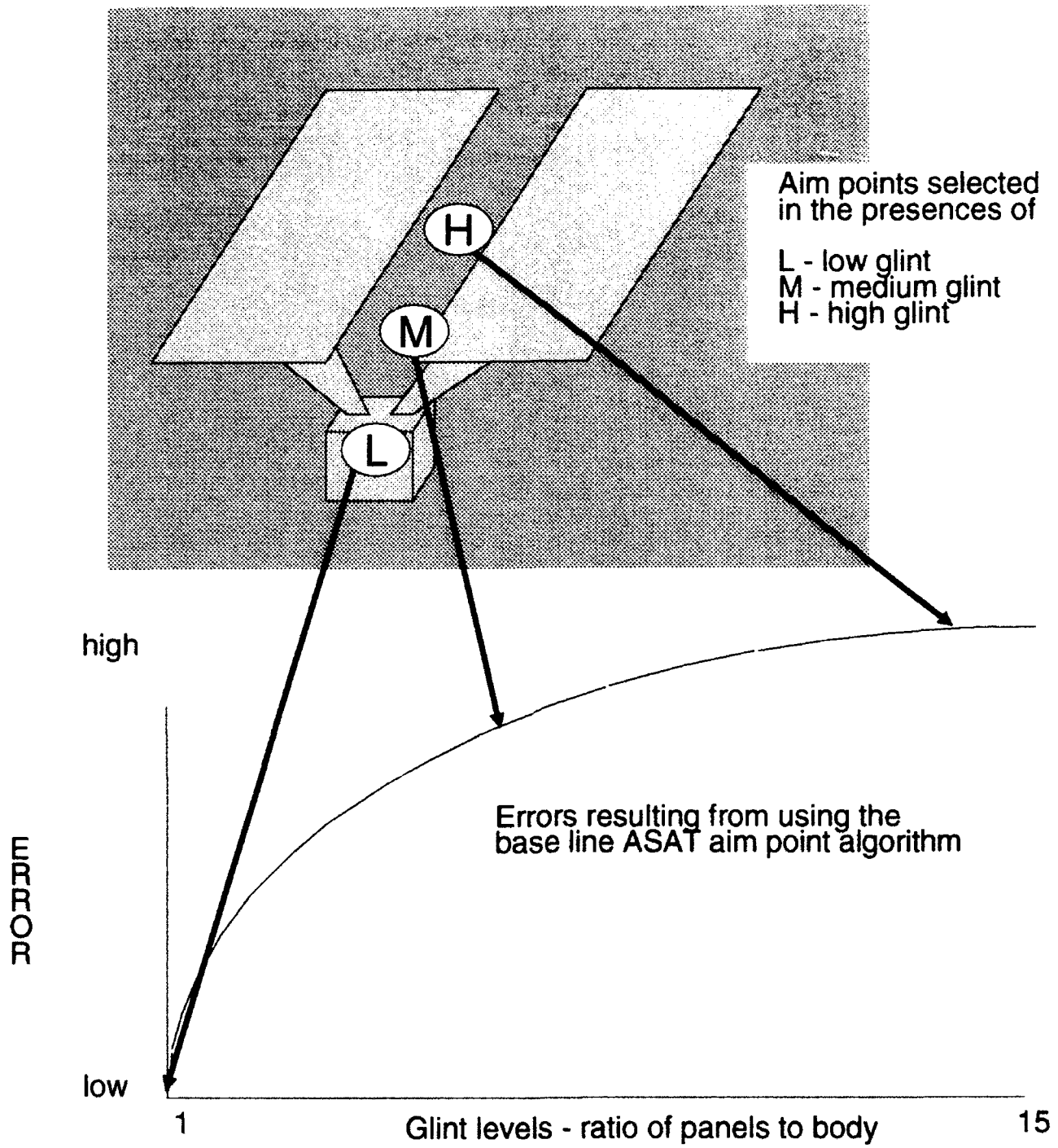


Figure 7: Justification for the range of glints used. The error approaches its maximum fairly quickly. In other words, it takes very little glint in order to cause the maximum amount of error due to that source.

algorithms identified relied exclusively on the neural network paradigm.

2.4.1 Adaptive Fuzzy Systems.

Several approaches to automatic fuzzy system development are based on neural network algorithms. This simply means that the learning features of neural networks are modified and applied to fuzzy systems. An example of this approach is described as adaptive fuzzy systems. Adaptive systems are systems which modify themselves in real-time to adapt to changing circumstances. This is actually more general then our requirements call for. Our application would simply call for applying a training set to the fuzzy system and letting it adapt until is produced satisfactory results.

REF: Kosko, B., *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, NJ, 1992.

Adaptive fuzzy systems have several problems. All approaches that we studied required an initial fuzzy system to be defined by the designer. This system was then modified in extremely limited ways during the adaptive process. Typically, weights would be assigned to each fuzzy rule. During the adaptive process the weights would be adjusted but the rules and membership functions would be unchanged.

A problem with most approaches based on neural networks is the need for training. Typically, a training set is used to train the system. This training process can be extremely time consuming. In addition, this process can also be unreliable based on the quality of the training set.

A further problem with neural network algorithms in general is that often it is difficult or impossible to interpret the results. The weights of the connections in the network usually do not correspond in any obvious way to the real world. In addition, the is no "paper trail" which allows replication or engineering review of the results.

2.4.2 Neural Networks.

Many of the approaches to automating fuzzy systems are based on neural networks. Almost invariably this means that the learning algorithm associated with some neural network is used to train a fuzzy system. More recently some researchers have developed neural networks which correspond exactly to a fuzzy system. This correspondence is a one-to-one mapping from the neural network to the fuzzy system. Again this approach requires starting with an initial neural network defined by the designer. Typically a back propagation algorithm will then be used to train the neural network and to produce the fuzzy system. This approach has the advantage that training can be used to define the membership functions as

well as weights on the fuzzy rules.

REF: S. Horikawa, T. Furuhashi, and Y. Uchikawa, "*On Fuzzy Modeling Using Fuzzy Neural Networks with the Back-Propagation Algorithm,*" IEEE Trans. Neural Networks, Vol 3, pp. 801-806.

2.4.3 Basis Functions.

In this approach, a system of basis functions are defined which can be combined to represent any membership function. Usually the basis functions are restricted to be gaussian but this is not necessary. The gaussian membership functions are then made to correspond to neural network activation functions (actually two sigmoidal activations functions represent one gaussian membership functions). The result is a neural network which can be trained by adjusting the slope and offset of these basis functions and adjusting various weights.

REF: L. Wang, and J. M. Mendel, "*Fuzzy Basis Functions, Universal Approximation, and Orthogonal Least-squares Learning,*" IEEE Trans. Neural Networks, Vol 3, pp.807-814, 1992.

2.4.4 Clustering Algorithms.

An approach to fuzzy rule generation is based on clustering algorithms. In this approach, hyper dimensional vectors represent the state of a system for several test cases. These vectors are then normalized onto the unit sphere. These normalized vectors are then clustered and each cluster is represented by a centroid vector. This set of centroid vectors then defines the fuzzy rules of the system.

REF: Kosko, B., *Neural Networks for Signal Processing,* Prentice Hall, Englewood Cliffs, NJ, 1992.

2.4.5 Computed Neural Networks.

Another approach is based on the idea of basis functions combined with some insight gained from our Phase I research. Our research has clearly shown that membership functions based on triangles are general enough to provide near optimal results in this application. In addition, it is almost certain that the rules and weights for a fuzzy system can be directly computed. This would provide the big advantage that training, which is extremely time consuming, would not be required. In addition, although training images would still be required, they could be generated by varying the parameters which cause the most significant errors.

2.5 Manual Rule Generation.

The manual rule generation task was initially one of the greatest areas of concern. We were worried that we would not be able to

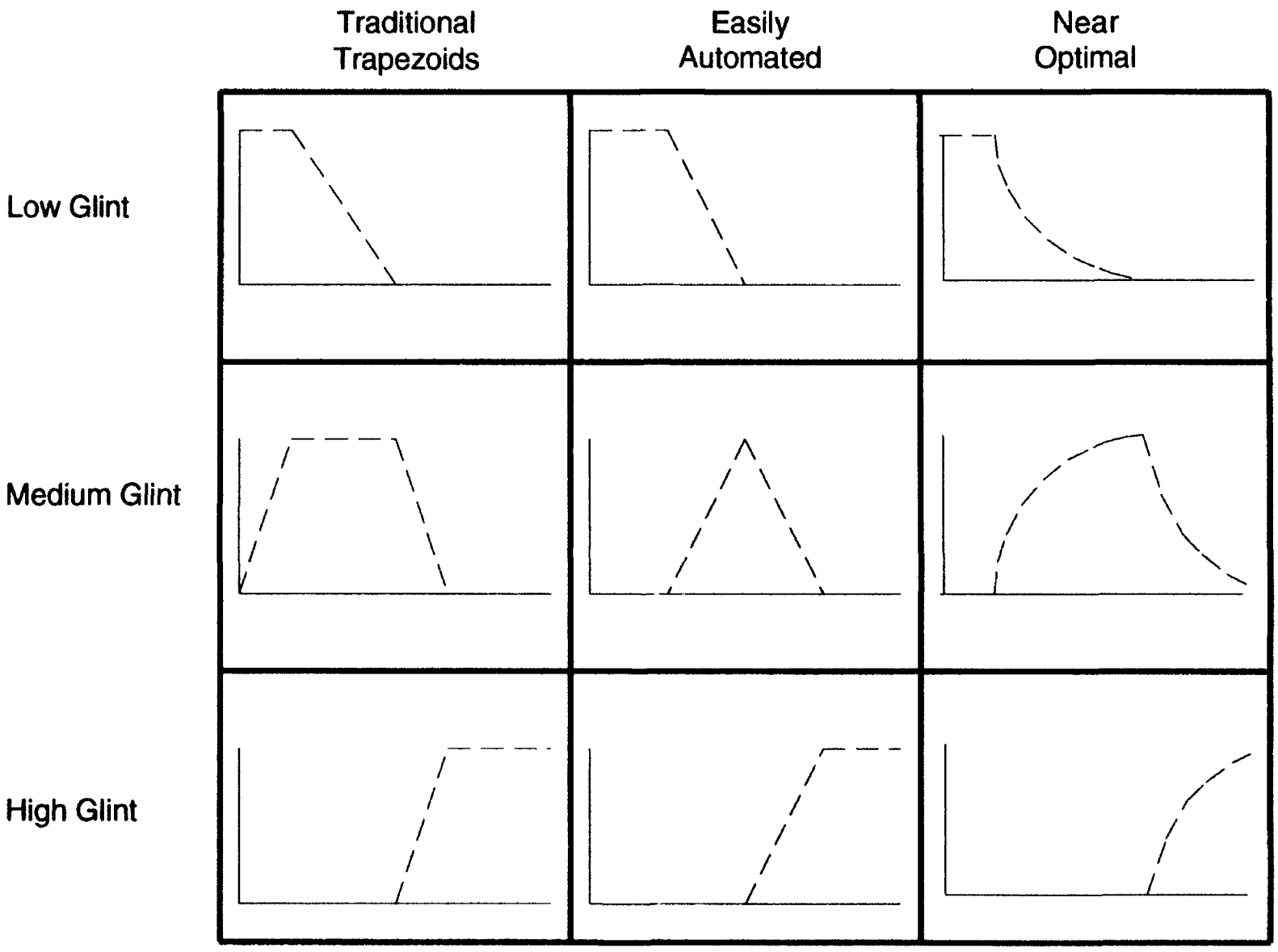


Figure 8: Three sets of three membership functions were computed. The Optimal membership function was determined based on the error curves of the three versions of the ASAT rules.

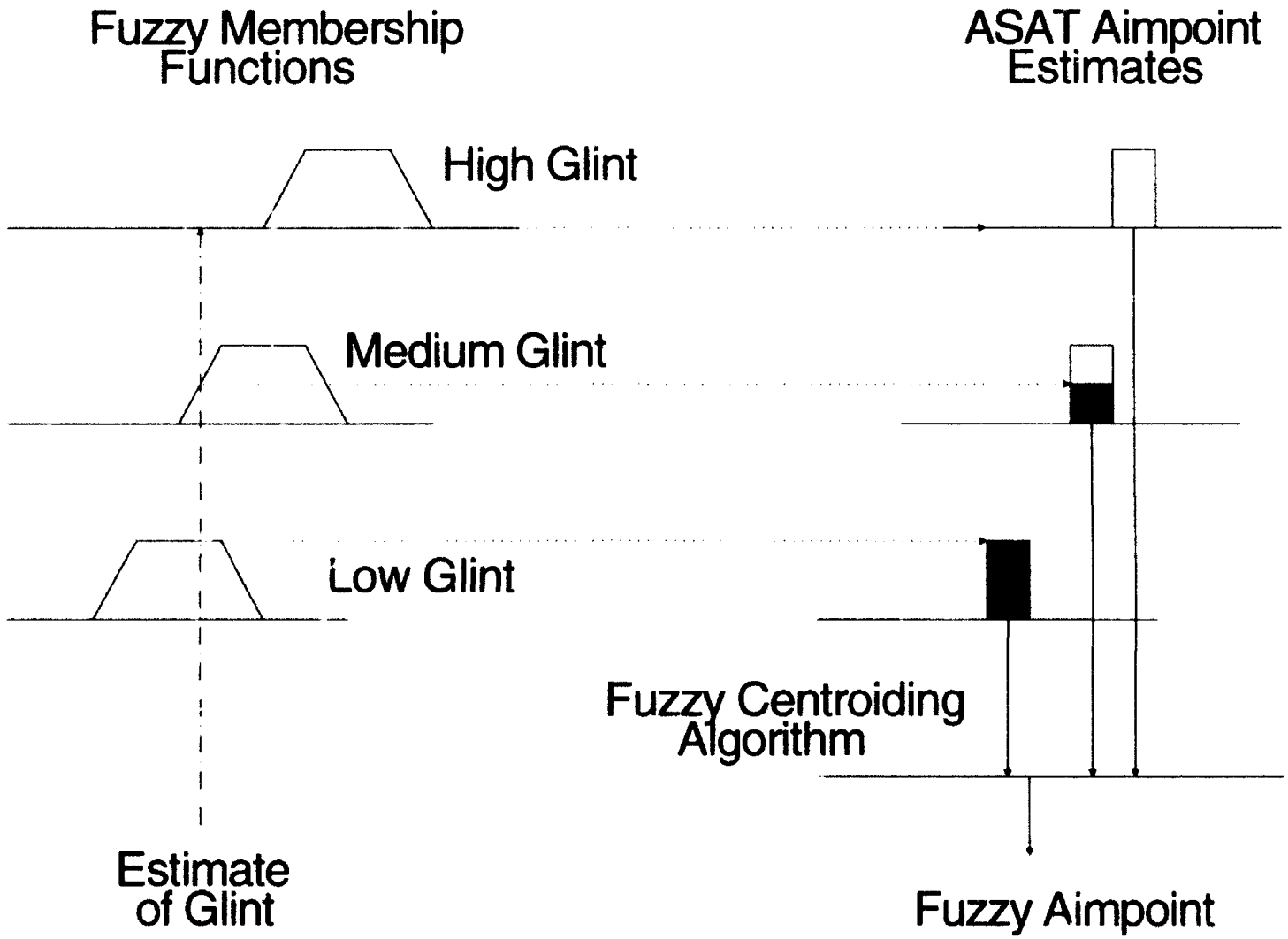
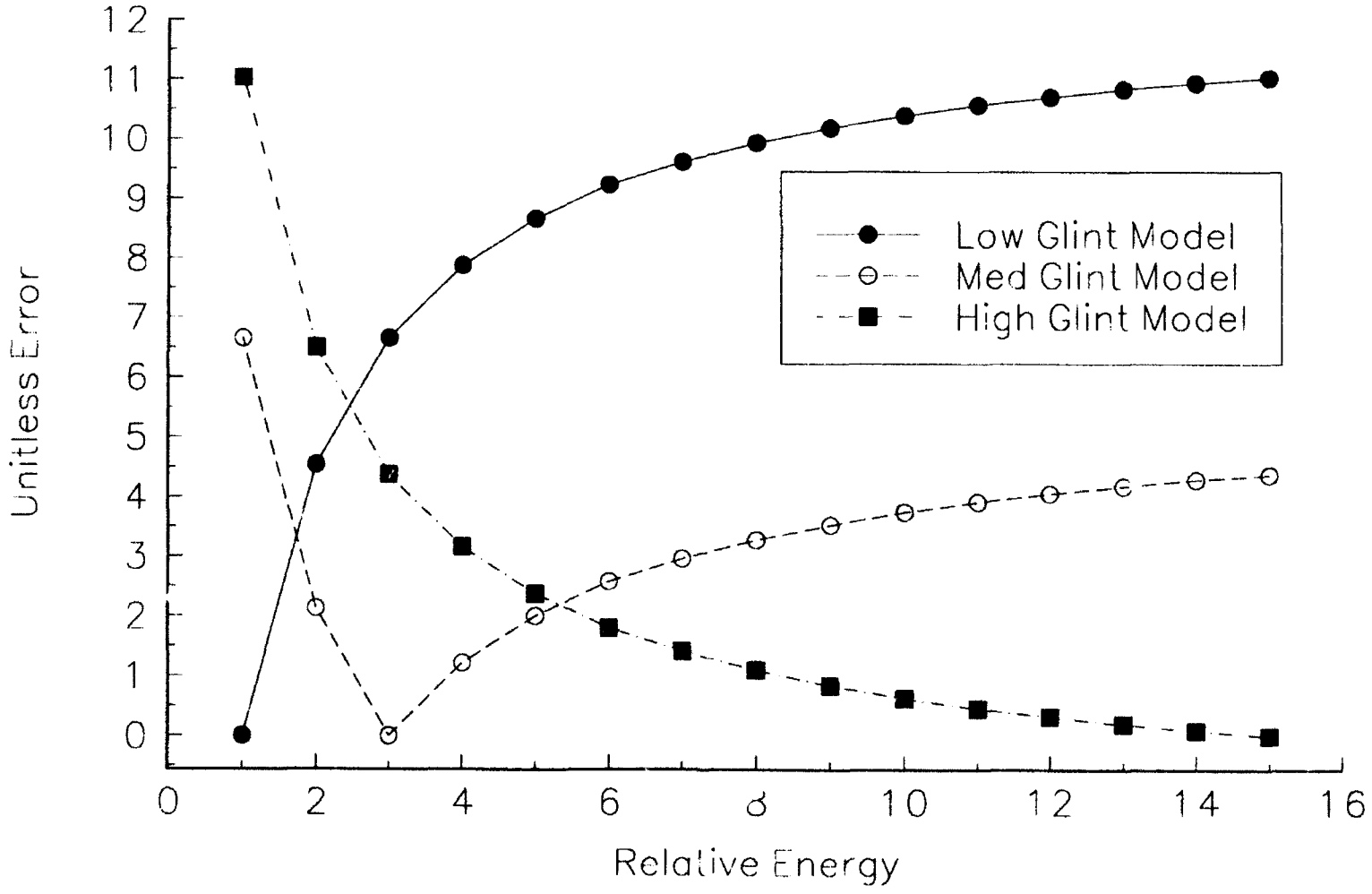


Figure 9: The membership functions determine how to weight the three versions of the ASAT rules. A simple centroiding scheme is used to defuzzify the answer and produce a final aim point.

Figure 10: Errors resulting from 3 models
based on low, medium, and high glint



manual generate rules and membership functions which were good enough. If the manual rules were not good enough, we would not have proven anything. Fortunately, fuzzy systems are robust enough that the sub-optimal rules and membership functions (see Figure 8) were nearly as effective as the what was later seen to be the optimal functions.

Our approach to generating the rules and membership functions provides an excellent model for an automated approach. We first assumed the simplest types of rules, namely if <condition> then <action> (see Figure 9). The results of all of these rules were then centroided to produce an aimpoint. The membership functions were defined by varying parameters of interest and then just looking at the surfaces produced (see Figure 10).

2.6 Manual Aimpoint Generation.

Initially it was felt that an human operator could provide a good estimate of the aimpoint by visual inspection. We designed a simple software tool which displays the image as squares of different shades of grey. The idea being that an operator could examine this image and extract an aimpoint. This approach failed for two reasons. First, the resolution of the image was very low - too low to allow an operator to find a reasonable aimpoint. In addition, there were only three shades of grey to work with - the visual effect was far from optimal.

2.7 Automatic Scoring.

Software was written to convert the aimpoints computed into errors. These errors were then processed through a MathCad program to produce graphs and to compute RMS values.

3.0 STATUS OF ACCOMPLISHMENTS

During this project, a study of the application of fuzzy logic to the ASAT aimpoint algorithm was performed. The purpose of this study was to determine if fuzzy logic could be effectively used to reduce the effects of various error sources - especially glint. At the completion of this study, we have produced a target simulation, applied fuzzy logic to the simulated images, and analyzed the results.

3.1 Develop target profiles and intensity maps.

This effort began with a definition of the planar surfaces of the target. From there a multi-stage software development effort resulted in programs to: generate a graphics image, translate the image into sub-pixels, and combine sub-pixels into pixels. During this process a point spread function was used to blur the image. Other sources of noise were ignored because it was assumed that the signal-to-noise ratio would be extremely high during the fina. phases of the mission. The result of this effort was software capable of supporting a parametric study of the fuzzy logic aimpoint algorithm.

3.2 Modify existing fuzzy logic software tool.

Several pieces of software already existed which were useful for this project.

For the purposes of comparison, it was necessary to implement the baseline ASAT aimpoint algorithms. Both the "centroid" and the "extent" algorithms were programmed and evaluated. The "extent" algorithm was found to be unacceptable for our application. As a consequence, only the "centroid" algorithm was carried forward for further evaluation.

The fuzzy logic engine was modified to accept the test image formats and was re-coded as a hardwired algorithm. This was necessary because of the huge amount of processing required for the image analysis and because of the large number of images generated and processed.

3.3 Develop display based scoring system.

This objective consisted of two parts: the display and the scoring system. The display was coded to allow an operator to view the images and attempt to select an aimpoint. Unfortunately, the resolution of the images is so low that it rapidly became clear that an operator was not competitive with even the ASAT baseline algorithms. This effort was abandoned with the resources redirected into the investigation of automated fuzzy system generation. The scoring system turned out to be a series of MATHCAD programs which analyzed and plotted the results.

3.4 Collect and analyze performance data.

The process of generating the simulated target images required almost a hundred hours of computing time. While this did not burn up much engineering time, it did limit the types of analysis that we could perform. Because of time constraints, it become necessary to abandon two equally interesting scenarios: glint off just one panel, and errors in the *a priori* estimate of the angle of attack. After a little analysis, it became obvious that these two scenarios are almost identical from the point of view of algorithms.

The results of the two panel glint study shows that with glint uniformly distributed over the range from no glint to maximal glint, there is approximately a factor of ten reduction in maximum aimpoint Error. Figures 11 and 12 show a substantial reduction in aimpoint errors for the near and medium range cases. As might be expected, the long range case shows considerable problems with both approaches (see Figure 13).

3.5 Future research and development.

Although fuzzy logic is not commonly used in the United States, it is very popular in other parts of the world. There is a large body of research in this area and many topics of potential future research have been identified. Two of these research topics are of particular interest: basis functions and hybrid fuzzy logic/neural network systems.

Basis functions appear to be an excellent way to approach automating the development of fuzzy systems. It appears to be relatively easy to compute basis functions automatically. These computed basis functions have been shown, in our Phase I study, to be reasonable substitutes for optimal basis functions.

Building hybrid fuzzy logic/neural network systems appear to have several advantages. These advantages include the possibility of avoiding the time consuming need to train neural networks. In addition, expert knowledge, which can easily be mapped into a fuzzy system, can now be mapped into neural networks.

Figure 11 : Close Range
Relative Errors

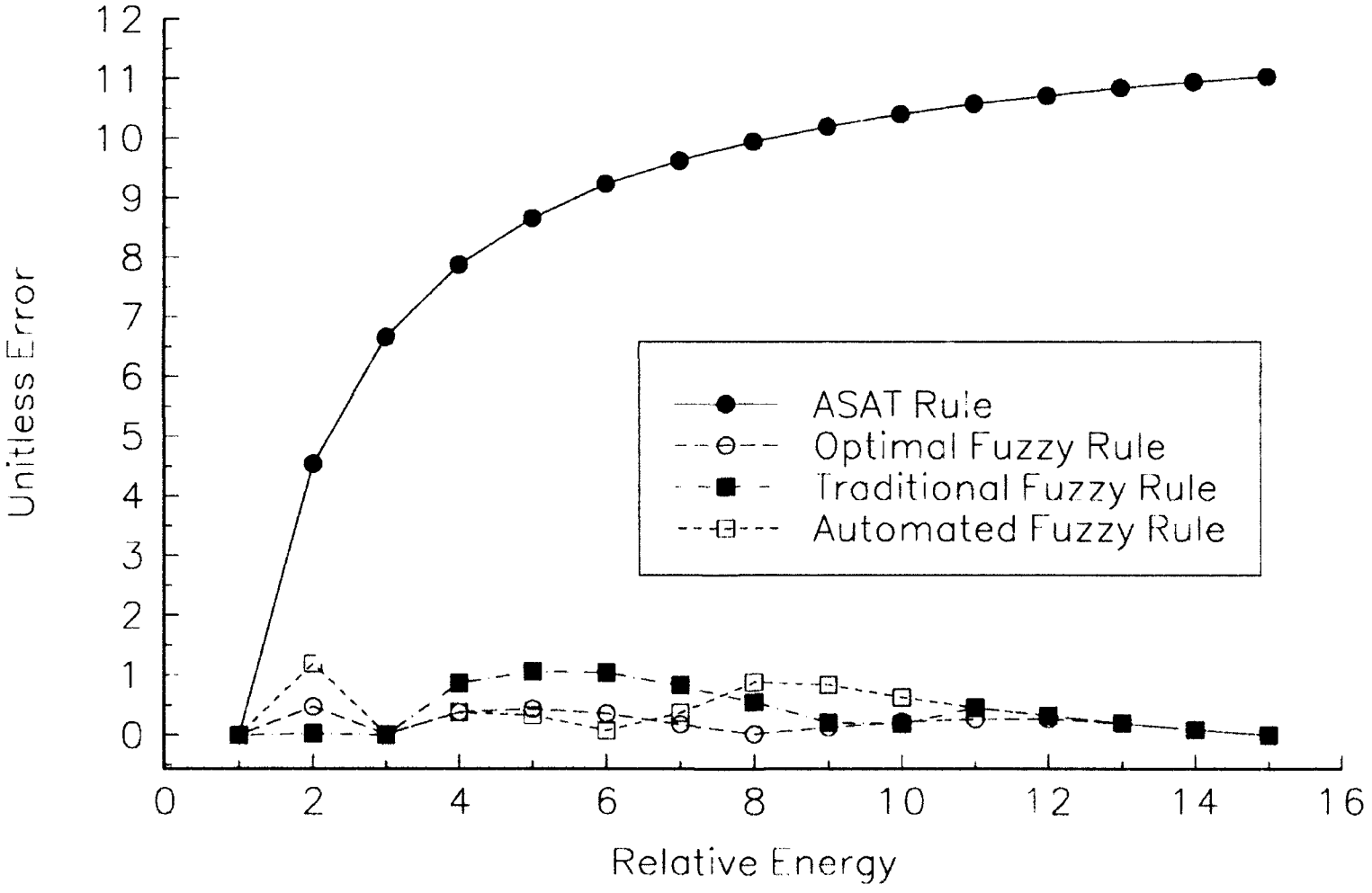


Figure 12: Mid Range
Relative Errors

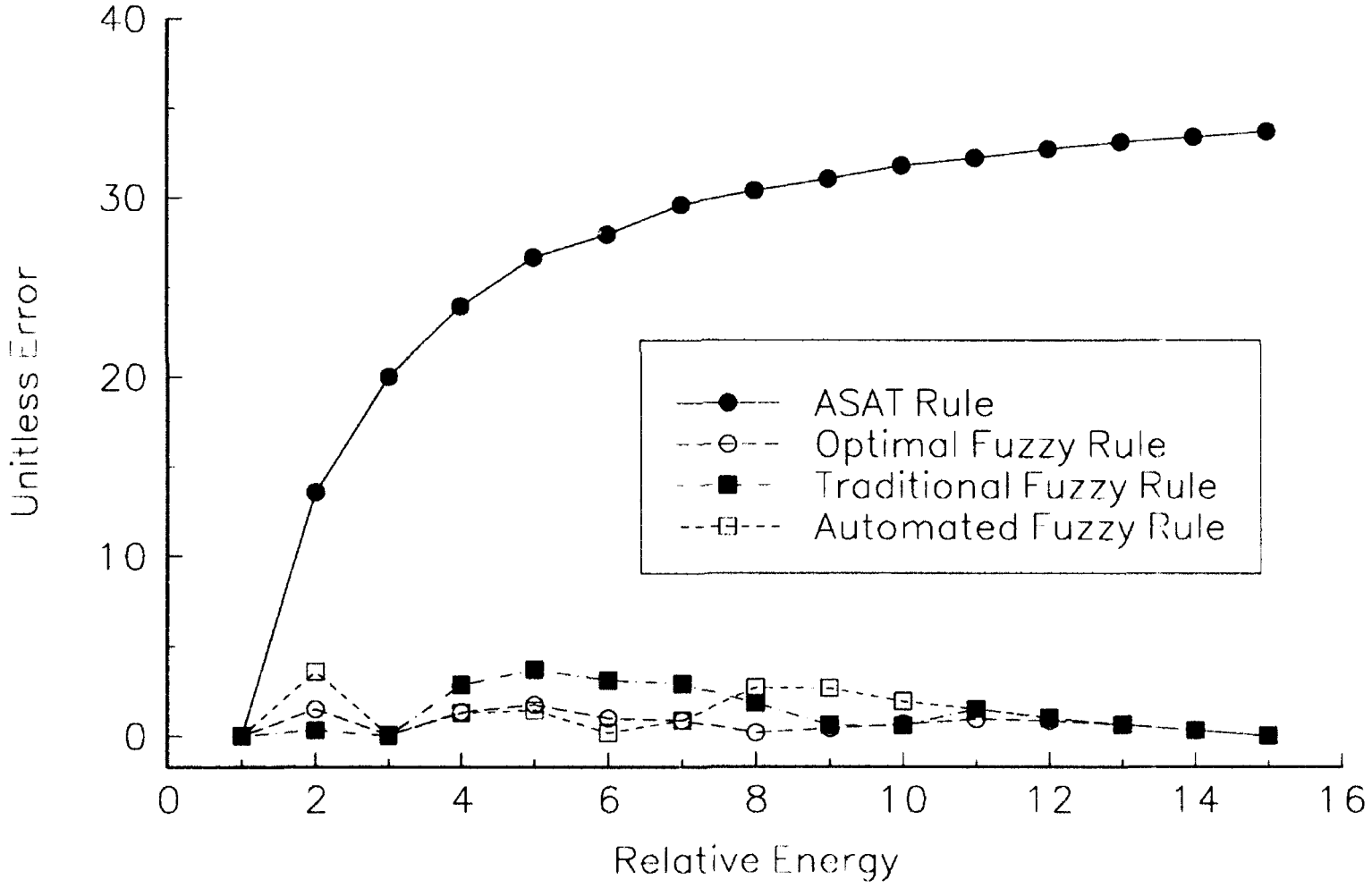
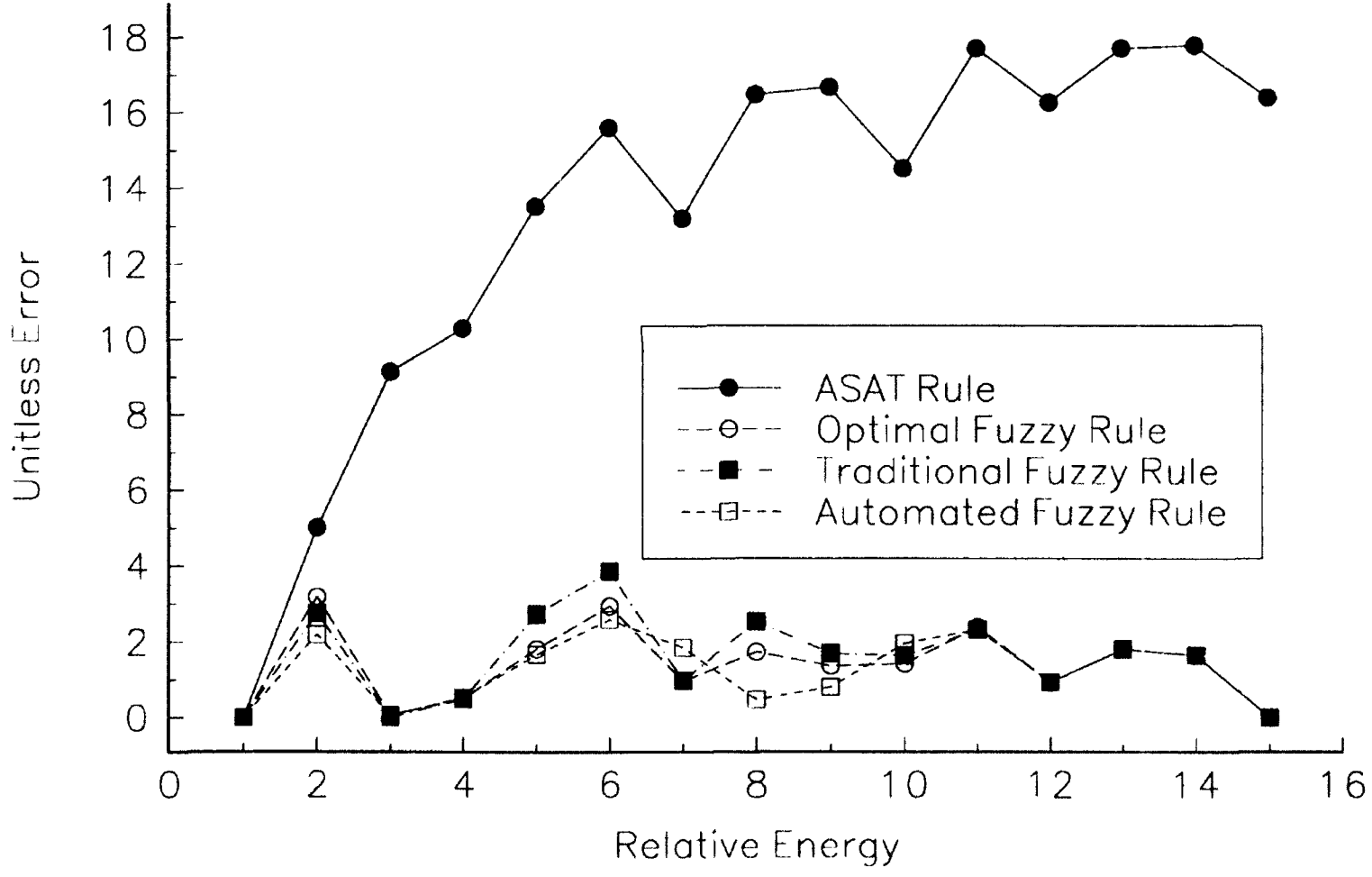


Figure 13: Far Range
Relative Errors



4.0 CONCLUSIONS AND RECOMMENDATIONS

4.1 Conclusions.

4.1.1 Fuzzy logic will reduce maximum error.

The current baseline ASAT aimpoint algorithms do not appear to be designed to handle glint or errors in the *a priori* estimates of angle of attack. Our analysis shows that these baseline algorithms can be modified to be incorporated into a fuzzy system. In addition, we have seen that this fuzzy system can reduce the maximum error in the aimpoint by a significant factor. This factor will of course depend on the target being attacked.

4.1.2 Fuzzy logic is robust.

Our research showed that fuzzy logic is robust in several different but important ways. While it is tolerant of uncertainties in the image under analysis, it is also tolerant of errors in the design of the fuzzy system. In this case, the error in the design was the use of sub-optimal membership functions.

4.1.3 Fuzzy logic can be fast.

Our preliminary estimates showed that the real-time part of the fuzzy system will be quite fast. This results from the fact that most of the processing would be performed just prior to launch.

4.1.4 Timely development of Fuzzy logic.

For near term projects (e.g. ASAT), it is vitally important to have the real-time software designed, coded, and tested in the near future. The real-time part of the fuzzy system code is simple enough that it could be designed quickly. In addition, the real-time code could also be designed first, before the non real-time code. This would allow a parallel development effort requiring only the interface definition.

4.1.5 Automating Fuzzy System Development.

It is possible to automate the development of fuzzy systems. Our analysis shows that, at least for this application, we will be able to effectively use certain properties of the underlying problem to great advantage. Not only are we able to use triangular membership functions but the membership functions are derived from simple multi-dimensional surfaces. This means that simple fuzzy rules can be combined to produce an extremely accurate aimpoint estimate.

4.1.6 Underlying algorithm needs to be continuous.

Fuzzy systems are continuous. This means that inputs that are close produce outputs that are close. One of the baseline ASAT aimpoint

algorithms was found to be discontinuous. This was unacceptable from our point of view and would probably by undesirable from the perspective of the ASAT Guidance and Control system.

4.1.7 Target needs to be blurred.

In order to achieve sub-pixel resolution it is necessary to have significant blurring. Clearly, if all of the target energy is contained within a single pixel, sub-pixel resolution is impossible. The problem with blurring is that it would reduce acquisition range. While this could possibly lead to a trade-off during the early design phase, at this point in the ASAT design the hardware is cast in concert.

4.2 Recommendations.

4.2.1 Need automated development system.

For many important military applications, and some commercial applications it will be necessary to automate the development of the fuzzy systems. During missions such as ASAT, there is insufficient time between selecting a target and launching an interceptor for the manual design of a fuzzy system. Of course this comes as no surprise because even the crisp logic rule based system development will need to be automated.

4.2.2 Errors in angle of attack.

Glint is widely recognized to be an important issue. Perhaps less widely recognized is the effects of errors in the *a priori* estimates of angle of attack. Our research shows that even a relatively small error in angle of attack can result in a serious reduction in probability of kill. For example, glint has been seen to cause significant errors. Fortunately, with these errors, the aimpoint is still contained within the target. Errors in the angle of attack, especially when combined with glint, on the other hand can cause the aimpoint to shift completely off of the target leading to a clean miss.

4.2.3 Computing neural networks.

One of the most serious problems associated with neural networks is that they need to be trained. Another problem is that once they are trained it is often impossible to analyze the connection weights and derive any understanding of the underlying problem or solution. Fuzzy logic can help solve both of these problems. Several researchers have developed one-to-one mappings between certain classes of fuzzy systems and neural networks. Since these two representations are equivalent, it becomes possible to use the benefits of both. Neural networks can be used to design fuzzy systems through training; fuzzy systems can be designed to directly compute connection weights for neural networks. Fuzzy systems have

the advantage that they are often extremely intuitive and therefore it could be relatively easy to determine the physical meaning of neural network connection weights by looking at them in a fuzzy context.

4.2.4 Investigate hybrid systems.

Fuzzy systems also have the advantage that they interface naturally with both neural networks and crisp logic rule-based systems. This suggests possibly forming an integrated package which includes an expert system for operator input, an neural network for complex analog data, and a fuzzy system to integrate or fuse the system.

5.0 POTENTIAL FOR COMMERCIALIZATION

There is a tremendous potential for commercializing fuzzy logic in the United States. In Japan, fuzzy logic has been in use for years, and it has been successfully applied to thousands of consumer products. Successful commercial applications include controlling water levels in washing machines, navigating river barges, and smoothly stopping high speed trains.

5.1 Selling aimpoint algorithms.

Aimpoint algorithms are important to several projects at the Strategic Defense Command e.g. GBI, THAADS, ERINT, DEW, E-I, PATRIOT, GSTS and LADAR. Many programs are having problems identifying aimpoint algorithms, and this appears to be a serious factor influencing probability of kill. In addition, many applications require very high degrees of accuracy in their aimpoint algorithms. The results of this study show a factor of ten reduction in maximum aimpoint errors resulting from the use of fuzzy logic.

5.2 Selling automated development tools.

Automatic tools for generating fuzzy systems could serve a critical need in the military. In the commercial world, automated tools could reduce the skill levels required of fuzzy system designers; this reduction in skill level would translate into potential increases in productivity and big savings. In addition, third party vendors (TOGAI InfraLogic) are interested in the possibility of combining their hardware with an automated fuzzy system development tool.

5.3 Selling fuzzy logic and neural network expertise.

Fuzzy logic is relatively unknown in the United States, and fuzzy system expertise is in demand. Since starting this contract, we have been asked to apply our fuzzy logic and neural network expertise to such problems as passive ranging (GBI), active damping control systems (BE), radar multipath (MTAS), and vehicle classification systems (DOT).

Appendix A - Software listings

```
/*
        filename: A_RULES
        This program accepts a pixel map and computes
        the ASAT aim point rules.
*/

# include <math.h>
# include <graph.h>
# include "worlddr.h"
# include <stdio.h>
# include "plot3d.h"
# include "hpplot.h"
# include "stdlib.h"

FILE    *pixelmap;                /* pixel map                */
FILE    *aptrules;                /* asat aimpoint rules      */
long    map[10][10];
char    name0[20];                 /* test file name           */
char    name1[20];                 /* pixel map file name      */
char    name2[20];                 /* asat aimpoint file name */
int     num_maps;
int     map_no;
int    api,apj;
int    aptop,apbottom;
int    apright,apleft;
long   maxsum,minsum;
int   top,bottom,right,left,temp;
float rhoc,rhor;
float cenc,cenr;
long   jsum,isum,tsum;
float jcen,icen;

/* Compute the ASAT intensity centroid aimpoint rules */

void Centroid(void)
{
int   jr,ic;

    isum = 0;
    jsum = 0;
    tsum = 0;
    for(jr=0;jr<10;jr++) {
        for(ic=0;ic<10;ic++) {
            temp = map[jr][ic];
            if(temp != 0) {
                jsum += temp*jr;
                isum += temp*ic;
                tsum += temp;
            }
        }
    }
    if(tsum > 0) {
```

```
        jcen = ((float)jsum)/((float)tsum);
        icen = ((float)isum)/((float)tsum);
    }
    else {
        jcen = -1.0;
        icen = -1.0;
    }

/* convert aimpoint rules to pixel coordinates */

    jcen = 22.0 + (jcen-2.0)*40.0;
    icen = 7.0  + (icen)    *80.0;


    cenc = ((float)api) - icen;
    cenr = ((float)apj) - jcen;
    fprintf(aptrules,"    cenc = %f   cenr = %f\n",cenc,cenr);
}

/* compute the ASAT extent aimpoint rules */

void Put_Up_box(void)
{
int   jr,ic;
    top = 10;
    bottom = -1;
    right = -1;
    left = 10;
    for(jr=0;jr<10;jr++) {
        for(ic=0;ic<10;ic++) {
            temp = map[jr][ic];
            if(temp != 0) {
                if(jr<top)    top =    jr,
                if(jr>bottom) bottom = jr;
                if(ic<left)   left =   ic;
                if(ic>right)  right =  ic;
            }
        }
    }

/* convert aimpoint rules to pixel coordinates */

    aptop =    22 + (top-2)     *40;
    apbottom = 21 + (bottom-1)  *40;
    apleft =    7 + left        *80;
    apright =   6 + (right+1)   *80;
    rhoc = ((float)(api - apleft))/((float)(apright - apleft));
    rhor = ((float)(apj - aptop))/((float)(apbottom - aptop));
    fprintf(aptrules,"\n    rhoc = %f   rhor = %f",rhoc,rhor);
}
```

```
/* read pixel map */

void Load_Map(void)
{
int   jr,ic;
    fscanf(pixelmap," %d map number",&map_no);
    fscanf(pixelmap," %d %d aimpoint",&api,&apj);
    for(jr=0;jr<10;jr++) {
        for(ic=0;ic<10;ic++) {
            fscanf(pixelmap," %d",&map[jr][ic]);
        }
        fscanf(pixelmap,"\n");
    }
    fscanf(pixelmap," %ld %ld maxsum and minsum",&maxsum,&minsum);
}
void main(int argc,char *argv[]){
int     im;
char   s[80];

/* read in file names */

    if(argc<3) {
        printf("This program requires a PMP and a ARL");
        printf(" file name in the command line.\n");
        getch();
        exit(0);
    }
    strcpy(name1,argv[1]);
    if((pixelmap = fopen(name1,"r")) == NULL) {
        printf("Could not open fine grain pixel map (%s)\n",name1);
        getch();
        exit(0);
    }
    strcpy(name2,argv[2]);
    if((aptrules = fopen(name2,"w")) == NULL) {
        printf("Could not open asat aimpoint rules (%s)\n",name2);
        getch();
        exit(0);
    }
    fscanf(pixelmap,"filename %s",s);
    fprintf(aptrules,"filename %s\n",name2);
    fscanf(pixelmap," %d number of maps",&num_maps);

/* compute ASAT aimpoint rules */

    for(im=0;im<num_maps;im++) {
        Load_Map();
        Put_Up_Box();
        Centroid();
    }
    fclose(pixelmap);
}
```

```
/*
        filename: C_FUZZY
        This program accepts a pixel map and computes
        the fuzzy aim points.
*/

# include <math.h>
# include <graph.h>
# include "worlddr.h"
# include <stdio.h>
# include "plot3d.h"
# include "hpplot.h"
# include "stdlib.h"

FILE    *pixelmap;                  /* pixel map                */
FILE    *lowrules;                  /* low aimpoint rules    */
FILE    *medrules;                  /* medium aimpoint rules    */
FILE    *highrules;                 /* high aimpoint rules    */
FILE    *fzyrules;                  /* fzy aimpoint rules    */
FILE    *aimpts;                    /* fuzzy aimpoints          */
long    map[10][10];
char    name0[20];                   /* test file name          */
char    name1[20];                   /* pixel map file name     */
char    name2[20];                   /* asat aimpoint rule name */
char    name3[20];                   /* asat aimpoint file name */
int     num_maps;
int     map_no;
int    api,apj;
int    aptop,apbottom;
int    apright,apleft;
long   maxsum,minsum;
int   top,bottom,right,left,temp;
float rhoc,rhor;
float lcenc,lcenr,mcenr,mcenc,hcenr,hcenc;
long   jsum,isum,tsum;
float jcen,icen;
float    fapi,fapj;
float    rho,rhot,lrho,mrho,hrho;

/* compute FUZZY aimpoint rules */

void Fuzzy(void)
{
int   jr,ic;
int   flag;
    isum = 0;
    jsum = 0;
    tsum = 0;
    for(jr=0;jr<10;jr++) {
        for(ic=0;ic<10;ic++) {
            temp = map[jr][ic];
            if(temp != 0) {
```

```
                    jsum += temp*jr;
                    isum += temp*ic;
                    tsum += temp;
                }
            }
        }
        if(tsum > 0) {
            jcen = ((float)jsum)/((float)tsum);
            icen = ((float)isum)/((float)tsum);
        }
        else {
            jcen = -1.0;
            icen = -1.0;
        }
        jcen = 22.0 + (jcen-2.0)*40.0;
        icen = 7.0  + (icen)    *80.0;

        rho =
    ((float)tsum-(float)minsum)/((float)maxsum-(float)minsum);
    /* apply fuzzy rule number "flag" */
        flag = 2;
        switch(flag) {
    case 0: /* traditional rules */
            if(rho<=0.0) {
                lrho = 1.0;
                mrho = 0.0;
                hrho = 0.0;
            }
            if((0.00<rho)&&(rho<=0.10)) {
                lrho = 1.0 - 10.0*rho;
                mrho =       10.0*rho;
                hrho = 0.0;
            }
            if((0.10<rho)&&(rho<=0.30)) {
                lrho = 0.0;
                mrho = 1.0;
                hrho = 0.0;
            }
            if((0.30<rho)&&(rho<=0.50)) {
                lrho = 0.0;
                mrho = 1.0 - 5.0*(rho-0.30);
                hrho =       5.0*(rho-0.30);
            }
            if(0.50<rho) {
                lrho = 0.0;
                mrho = 0.0;
                hrho = 1.0;
            }
            break;
    case 1: /* optimal rule */
            if(rho<=0.0) {
                lrho = 1.0;
```

```
            mrho = 0.0;
            hrho = 0.0;
        }
        if((0.00<rho)&&(rho<=0.25)) {
            lrho = 1.0 - 4.0*rho;
            mrho =       4.0*rho;
            hrho = 0.0;
        }
        if((0.25<rho)&&(rho<=0.50)) {
            lrho = 0.0;
            mrho = 1.0 - 4.0*(rho-0.25);
            hrho =       4.0*(rho-0.25);
        }
        if(0.50<rho) {
            lrho = 0.0;
            mrho = 0.0;
            hrho = 1.0;
        }
        break;
case 2: /* automated rule */
        if(rho<=0.0) {
            lrho = 1.0;
            mrho = 0.0;
            hrho = 0.0;
        }
        if((0.00<rho)&&(rho<=0.30)) {
            rhot = (0.30 - rho)*(0.30 - rho)/0.09;
            lrho = rhot;
            mrho = 1.0 - rhot;
            hrho = 0.0;
        }
        if((0.30<rho)&&(rho<=0.50)) {
            rhot = (0.50 - rho)*(0.50 - rho)/0.04;
            lrho = 0.0;
            mrho = rhot;
            hrho = 1.0 - rhot;
        }
        if(0.50<rho) {
            lrho = 0.0;
            mrho = 0.0;
            hrho = 1.0;
        }
        break;
    }

    fapi = icen + lrho*lcenc+mrho*mcenc+hrho*hcenc;
    fapj = jcen + lrho*lcenr+mrho*mcenr+hrho*hcenr;
    fprintf(aimpts," %d   %d   %f   %f  \n\n",api,apj,fapi,fapj);
}
/* load pixel map */
void Load_Map(void)
{
```

```
int   jr,ic;

    fscanf(lowrules," rhoc = %f rhor = %f cenc = %f cenr = %f",
                    &rhoc,&rhor,&lcenc,&lcenr);
    fscanf(medrules," rhoc = %f rhor = %f cenc = %f cenr = %f",
                    &rhoc,&rhor,&mcenc,&mcenr);
    fscanf(highrules," rhoc = %f rhor = %f cenc = %f cenr = %f",
                    &rhoc,&rhor,&hcenc,&hcenr);
    fscanf(pixelmap," %d map number",&map_no);
    fscanf(pixelmap," %d %d aimpoint",&api,&apj);
    for(jr=0;jr<10;jr++) {
        for(ic=0;ic<10;ic++) {
            fscanf(pixelmap," %d",&map[jr][ic]);
        }
        fscanf(pixelmap,"\n");
    }
    fscanf(pixelmap," %ld %ld maxsum and minsum",&maxsum,&minsum);

}
void main(int argc,char *argv[]){
int     im;
char   s[80]; /* read in file names */
    if(argc<7) {
printf("This program requires an FRL, PMP, FPT and a lmh APT");
        printf(" file name in the command line.\n");
        getch();
        exit(0);
    }
    strcpy(name1,argv[1]);
    if((fzyrules = fopen(name1,"r")) == NULL) {
printf("Could not open the fuzzy aimpoint rules (%s)\n",name1);
        getch();
        exit(0);
    }
    strcpy(name2,argv[2]);
    if((pixelmap = fopen(name2,"r")) == NULL) {
        printf("Could not open test  pixel map (%s)\n",name2);
        getch();
        exit(0);
    }
    strcpy(name3,argv[3]);
    if((aimpts = fopen(name3,"w")) == NULL) {
        printf("Could not open fuzzy aimpoint file (%s)\n",name3);
        getch();
        exit(0);
    }
    strcpy(name3,argv[4]);
    if((lowrules = fopen(name3,"r")) == NULL) {
printf("Could not open lo asat aimpoint file (%s)\n",name3);
        getch();
        exit(0);
    }
```

```
    strcpy(name3,argv[5]);
    if((medrules = fopen(name3,"r")) == NULL) {
printf("Could not open med asat aimpoint file (%s)\n",name3);
        getch();
        exit(0);
    }
    strcpy(name3,argv[6]);
    if((highrules = fopen(name3,"r")) == NULL) {
printf("Could not open hi asat aimpoint file (%s)\n",name3);
        getch();
        exit(0);
    }
    fprintf(aimpts,"filename %s\n",name3);
    fscanf(pixelmap,"filename %s",s);
    fscanf(lowrules,"filename %s",s);
    fscanf(medrules,"filename %s",s);
    fscanf(highrules,"filename %s",s);
    fscanf(pixelmap," %d number of maps",&num_maps);
    fprintf(aimpts,"\n  %d number of maps\n\n",num_maps);
/* compute fuzzy aimpoint rules */
    for(im=0;im<num_maps;im++) {
        Load_Map();
        Fuzzy();
    }
    fclose(pixelmap);
}
```

```
/*
        filename: DISP
        This program accepts a test pixel map and displays it.
*/

# include <math.h>
# include <graph.h>
# include "worlddr.h"
# include <stdio.h>
# include "plot3d.h"
# include "hpplot.h"
# include "stdlib.h"

FILE    *pixelmap;
char    name0[20];
char    name1[20];
long     map[10][10];
char  title[80];
int   num_maps;
int   map_no;
int    api,apj;
long   maxsum,minsum;
/* display pixel map */
void DisplayMap(void)
{
int    i,j,i1,j1;
int    row,col,color;
float  temp;
int    temp1,temp2,temp3;
int    colornum;
    for(j=0;j<10;j++) {
        for(i=0;i<10;i++) {
            temp = (float) map[j][i];
            temp = temp/3200.0;
            colornum = 1;
temp1 = (int) ((0.34 - 0.33*(temp - 5.0)/5.0)*(32767.0));
temp2 = (int) ((1.00 - 0.33*(temp - 5.0)/5.0)*(32767.0));
            if(temp<=5.0) {
                colornum = 0;
                temp1 = (int) ((1.0 - 0.66*temp/5.0)*(32767.0));
            }
            else {
                if (temp>=10.) {
                    colornum = 2;
temp2 = (int) ((0.33 + 0.66*(temp - 10.0)/5.0)*(32767.0));
                }
            }
/* color each pixel */
            row = i*56;
            for(i1=0;i1<56;i1++) {
                row++;
                col = j*28+15;
```

```
                    for(j1=0;j1<28;j1++) {
                        col++;
                        temp3 = rand(); /* select random mix of */
                              /* colors to represent intensity */
                        switch(colornum) {
            case 0:
                            if(temp3<temp1) _setcolor(0);
                            else            _setcolor(7);
                            break;
            case 1:
                            if(temp3<temp1)     _setcolor(0);
                            else {
                                if(temp3>temp2) _setcolor(15);
                                else            _setcolor(7);
                            }
                            break;
            case 2:
                            if(temp3<temp2) _setcolor(15);
                            else            _setcolor(7);
                            break;
                        }
                        _setpixel(row,col);
                    }
                }
            }
        }
    }
    void LoadMap(void)
    { /* read in pixel map */
    int   jr,ic;

        fscanf(pixelmap," %d map number",&map_no);
        fscanf(pixelmap," %d %d aimpoint",&api,&apj);
        for(jr=0;jr<10;jr++) {
            for(ic=0;ic<10;ic++) {
                fscanf(pixelmap," %d",&map[jr][ic]);
            }
            fscanf(pixelmap,"\n");
        }
        fscanf(pixelmap," %ld %ld maxsum and minsum",&maxsum,&minsum);
    }
    void main(int argc,char *argv[]) {
    int     im;
    char   s[80]; /* read in file names */
        if(argc<2) {
            printf("This program requires a test");
            printf(" file name in the command line\n");
            getch();
            exit(0);
        }
        strcpy(name1,argv[1]);
        if((pixelmap = fopen(name1,"r")) == NULL) {
```

```
            printf("Could not open pixel map (%s)\n",name1);
            getch();
            exit(0);
        }
        _setvideomode(_ERESCOLOR);
        fscanf(pixelmap,"filename %s",title);
        fscanf(pixelmap," %d number of maps",&num_maps);
        for(im=0;im<num_maps;im++) { /* display images */
            LoadMap();
            DisplayMap();
            getch();
            _clearscreen(_GVIEWPORT);
        }
        CloseSEGraphics();
        fclose(pixelmap);
    }
```

```
/*
        filename: MK_ASAT
        This program accepts a pixel map and computes
        the ASAT aim points and errors for plotting.
*/

# include <math.h>
# include <graph.h>
# include "worlddr.h"
# include <stdio.h>
# include "plot3d.h"
# include "hpplot.h"
# include "stdlib.h"

FILE    *pixelmap;                  /* pixel map                */
FILE    *aptrules;                  /* asat aimpoint rules      */
FILE    *allpts;                    /* all aimpoints            */
long    map[10][10];
char    name0[80];                   /* test file name           */
char    name1[80];                   /* pixel map file name      */
char    name2[80];                   /* asat aimpoint rule name  */
char    name3[80];                   /* asat aimpoint file name  */
char    name4[20];                   /* all aimpoint file name   */
int     num_maps;
int     map_no;
int    api,apj;
int    aptop,apbottom;
int    apright,apleft;
long   maxsum,minsum;
int   top,bottom,right,left,temp;
float rhoc,rhor,cenc,cenr;
long   jsum,isum,tsum;
float jcen,icen;
float    fapi,fapj;
float   erout[10];
float  jbc,ibr,jcc,icr,erb,erc;
double dcc,dcr,dpc,dpr,dnm;
double dtemp1,dtemp2,dtemp3;
/* compute the intensity centroid and compute aimpoint */
void Centroid(int num_maps,int im)
{
int   jr,ic;

    isum = 0;
    jsum = 0;
    tsum = 0;
    for(jr=0;jr<10;jr++) {
        for(ic=0;ic<10;ic++) {
            temp = map[jr][ic];
            if(temp != 0) {
                jsum += temp*jr;
                isum += temp*ic;
```

```
                    tsum += temp;
                }
            }
        }
        if(tsum > 0) {
            jcen = ((float)jsum)/((float)tsum);
            icen = ((float)isum)/((float)tsum);
        }
        else {
            jcen = -1.0;
            icen = -1.0;
        } /* convert to pixel coordinates */
        jcen = 22.0 + (jcen-2.0)*40.0;
        icen = 7.0  + (icen)    *80.0;

        dcc = icen + cenc;
        dcr = jcen + cenr;
        dnm = (float) (num_maps - im);
        dpc = (double) api;
        dpr = (double) apj;
        dtemp1 = pow((dpr-dcr)*dnm,2.0);
        dtemp2 = pow((dpc-dcc)*dnm,2.0);
        dtemp3 = dtemp1 + dtemp2;
        erc = (float) pow(dtemp3,0.5);
        erout[im] = erc;
    }
    void Load_Map(void)
    { /* read pixel map */
    int   jr,ic;

        fscanf(aptrules," rhoc = %f rhor = %f cenc = %f cenr = %f",
                        &rhoc,&rhor,&cenc,&cenr);
        fscanf(pixelmap," %d map number",&map_no);
        fscanf(pixelmap," %d %d aimpoint",&api,&apj);
        for(jr=0;jr<10;jr++) {
            for(ic=0;ic<10;ic++) {
                fscanf(pixelmap," %d",&map[jr][ic]);
            }
            fscanf(pixelmap,"\n");
        }
        fscanf(pixelmap," %ld %ld maxsum and minsum",&maxsum,&minsum);
    }
    void main(int argc,char *argv[]){
    int     im; /* read in file names */
    char   s[80];
        if(argc<3) {
            printf("This program requires an ARL, and PMP");
            printf(" file name in the command line.\n");
            getch();
            exit(0);
        }
        strcpy(name1,argv[1]);
```

```
    if((aptrules = fopen(name1,"r")) == NULL) {
printf("Could not open the asat aimpoint rules (%s)\n",name1);
        getch();
        exit(0);
    }
    strcpy(name2,argv[2]);
    if((pixelmap = fopen(name2,"r")) == NULL) {
        printf("Could not open test  pixel map (%s)\n",name2);
        getch();
        exit(0);
    }
    if((allpts = fopen("g:temp2.tmp","w")) == NULL) {
        printf("Could not open temporary file\n");
        getch();
        exit(0);
    }
    fscanf(pixelmap,"filename %s",s);
    fscanf(aptrules,"filename %s",s);

    fscanf(pixelmap," %d number of maps",&num_maps);
    for(im=0;im<num_maps;im++) { /* compute ASAT aimpoint errors */
        Load_Map();
        Centroid(num_maps,im);
    }
    for(im=num_maps-1;im>=0;im--) {
        fprintf(allpts,"    %f",erout[im]);
    }
    fprintf(allpts,"\n\n");
}
```

```
/*
        filename: MK_FRN
        This program accepts the fuzzy aimpoint lists,
        computes the errors and formats the data
        for plotting with mathcad.
*/

# include <math.h>
# include <graph.h>
# include "worlddr.h"
# include <stdio.h>
# include "plot3d.h"
# include "hpplot.h"
# include "stdlib.h"

FILE    *aimpts;                /* asat aimpoints          */
FILE    *fzypts;                /* fuzzy aimpoints          */
FILE    *manpts;                /* manual aimpoints          */
FILE    *allpts;                /* all aimpoints          */
long    map[10][10];
char    name0[20];               /* test file name          */
char    name1[20];               /* asat aimpoint file name */
char    name2[20];               /* fuzzy aimpoint file name */
char    name3[20];               /* manual aimpoint file name */
char    name4[20];               /* all aimpoint file name */
int     num_maps;
int     map_no;
float   erout[20];
float  jbc,ibr,jcc,icr,erb,erc;
double dcc,dcr,dpc,dpr,dnm;
double dtemp1,dtemp2,dtemp3;

void ComputeError(int maps,int im)
{ /* compute fuzzy aimpoint errors */
int    jpc,ipr;

    dnm = (float) (maps - im);

    fscanf(aimpts," %d %d %f %f",&jpc,&ipr,&jcc,&icr);
    dcc = (double) jcc;
    dpc = (double) jpc;
    dcr = (double) icr;
    dpr = (double) ipr;
    dtemp1 = pow((dpr-dcr)*dnm,2.0);
    dtemp2 = pow((dpc-dcc)*dnm,2.0);
    dtemp3 = dtemp1 + dtemp2;
    erc = (float) pow(dtemp3,0.5);
    erout[im] = erc;

}
void main(int argc,char *argv[]){
int     im; /* read in file names */
```

```
char   s[80];
    if(argc<2) {
        printf("This program requires an FPT file in");
        printf(" the command line.\n");
        getch();
        exit(0);
    }
    strcpy(name0,argv[1]);
    if((aimpts = fopen(name0,"r")) == NULL) {
        printf("Could not open fuzzy aimpoints file (%s)\n",name0);
        getch();
        exit(0);
    }
    if((allpts = fopen("g:temp2.tmp","w")) == NULL) {
        printf("Could not open temporary file\n');
        getch();
        exit(0);
    }
    fscanf(aimpts,"filename %s",name1);
    fscanf(aimpts," %d number of maps",&num_maps);
    for(im=0;im<num_maps;im++) {
        ComputeError(num_maps,im);
    }
    for(im=num_maps-1;im>=0;im--) {
        fprintf(allpts,"    %f",erout[im]);
    }
    fprintf(allpts,"\n\n");
    fflush(aimpts);
    fclose(aimpts);
}
```

```
/*
        filename: MK_MAP
        This program accepts a test target description
        file and outputs test pixel maps.
*/
#define   MAX_IMAGES   100
#define   MAX_SURFACES 10
# include <math.h>
# include <graph.h>
# include "worlddr.h"
# include <stdio.h>
# include "plot3d.h"
# include "hpplot.h"

int       err,i,j;
int       color;
struct    WorldRect wr;

float    x,y,z;
float      rot[MAX_IMAGES],xrot,yrot,zrot;
int      axn[MAX_IMAGES],colorno[MAX_SURFACES][MAX_IMAGES];
int      fcnt[MAX_IMAGES];
float    p[4][3][MAX_SURFACES];
float    apx,apy,apz;
int      api,apj;
int      no_images,no_surfaces,no_frames;
int      old_f_cnt,frame_cnt;
long     tempsum[56][56];
float     tempspread[80][80];
FILE     *tdffile;
FILE     *pixelmap;
FILE     *noisemodel;
char     name0[20];
char     name1[20];
char     name2[20];
char     name3[20];
float    coefs[3];

void DrawAndSaveAimPoint(void) {
struct  point3D pv[5];
int i,j; /* draw aimpoint of target */
    pv[0].x = apx;
    pv[0].y = apy;
    pv[0].z = apz;
    pv[1].x = apx + 0.05;
    pv[1].y = apy;
    pv[1].z = apz;
    pv[2].x = apx + 0.05;
    pv[2].y = apy + 0.05;
    pv[2].z = apz;
    pv[3].x = apx;
    pv[3].y = apy + 0.05;
```

```
    pv[3].z = apz;
    pv[4].x = apx;
    pv[4].y = apy;
    pv[4].z = apz;
    SelectColor(1);
    PolyFill3D(pv,1,1,5);
    api = -1;
    apj = -1;
    for(i=0;i<560;i++) {
        for(j=0;j<280;j++) {
            if(_getpixel(i,j)==1) {
                api = i;
                apj = j;
                break;
            }
        }
        if(api!=-1) break;
    }
}
void DrawTarget(int image_no) {
struct  point3D pv[5];
int     isf,ipt;
/* draw image of target */
    for(isf=0;isf<no_surfaces;isf++) {
        for(ipt=0;ipt<4;ipt++) {
            pv[ipt].x = p[ipt][0][isf];
            pv[ipt].y = p[ipt][1][isf];
            pv[ipt].z = p[ipt][2][isf];
        }
        pv[4].x = p[0][0][isf];
        pv[4].y = p[0][1][isf];
        pv[4].z = p[0][2][isf];

        SelectColor(colorno[isf][image_no]);
        PolyFill3D(pv,1,colorno[isf][image_no],5);
    }
}
void LoadTargets(void)
{ /* read target description file */
int   im,is,ip;
int   image,surface;

    fscanf(tdffile," %d number of surfaces",&no_surfaces);

    for(is=0;is<no_surfaces;is++) {
        for(ip=0;ip<4;ip++) {
            fscanf(tdffile," %f %f %f",
            &p[ip][0][is],&p[ip][1][is],&p[ip][2][is]);
        }
    }

    fscanf(tdffile," %f %f %f aimpoint",&apx,&apy,&apz);
```

```
    fscanf(tdffile," %f %f %f xrot yrot zrot",&xrot,&yrot,&zrot);
    WorldRotate3(xrot,0);
    WorldRotate3(yrot,1);
    WorldRotate3(zrot,2);

    fscanf(tdffile," %d number of frames",&no_frames);

    fprintf(pixelmap,"\n%d number of maps\n",no_frames);

    fscanf(tdffile," %d number of images",&no_images);

    for(im=0;im<no_images;im++) {
        fscanf(tdffile," %d %f %d %d image rotation axes fcount",
                    &image,&rot[im],&axn[im],&fcnt[im]);
        for(is=0;is<no_surfaces;is++) {
fscanf(tdffile," %d %d surface color",&surface,&colorno[is][im]);
        }
    }
}
void PSF(int r1,int c1,int r2,int c2,float coef)
{ /* point spread function */
    if((r2>=0)&&(r2<56)&&(c2>=0)&&(c2<56))
        tempspread[r1][c1]+=coef*((float) tempsum[r2][c2]);
}

void SaveMap(int image_no)
{ /* compute and save pixel map */
int   ic01,jr01,ic02,jr02,ic03,jr03;
long  sum,maxsum,minsum;
int   jrow01,icol01,jrow02,icol02;
int   row,col,pix;
    fprintf(pixelmap,"\n%d map number\n",image_no);
    fprintf(pixelmap,"\n%d %d aimpoint\n\n",api,apj);
    maxsum = 0;
    minsum = 0;
    for(jr01=0;jr01<7;jr01++) {
        for(ic01=0;ic01<7;ic01++) {
            for(ic02=0;ic02<8;ic02++) {
                for(jr02=0;jr02<8;jr02++) {
                    sum = 0;
                    for(ic03=0;ic03<10;ic03++) {
                        for(jr03=0;jr03<5;jr03++) {
                            col = (ic01*8+ic02)*10+ic03+0   +7;
                            row = (jr01*8+jr02)*5 +jr03+15  +7;
                            pix = _getpixel(col,row);
                            if(pix > 0) {
                                sum += pix;
                                maxsum += 15;
                                minsum += 1;
                            }
                        }
                    }
```

```
                        tempsum[jr01*8+jr02][ic01*8+ic02] = sum;
                    }
                }
            }
        }
        for(jr01=0;jr01<10;jr01++) {
            for(ic01=0;ic01<10;ic01++) {
                for(ic02=0;ic02<8;ic02++) {
                    for(jr02=0;jr02<8;jr02++) {
                        icol01 = ic01*8+ic02;
                        jrow01 = jr01*8+jr02;
                        tempspread[jrow01][icol01] = 0;
                        for(ic03=-4;ic03<4;ic03++) {
                            for(jr03=-4;jr03<4;jr03++) {
                                jrow02 = jrow01-16+jr03;
                                icol02 = icol01+ic03;
PSF(jrow01,icol01,jrow02  ,icol02  ,coefs[0]);
PSF(jrow01,icol01,jrow02-8,icol02  ,coefs[1]);
PSF(jrow01,icol01,jrow02,icol02-8  ,coefs[1]);
PSF(jrow01,icol01,jrow02+8,icol02  ,coefs[1]);
PSF(jrow01,icol01,jrow02,icol02+8  ,coefs[1]);
PSF(jrow01,icol01,jrow02-8,icol02-8,coefs[2]);
PSF(jrow01,icol01,jrow02-8,icol02+8,coefs[2]);
PSF(jrow01,icol01,jrow02+8,icol02-8,coefs[2]);
PSF(jrow01,icol01,jrow02+8,icol02+8,coefs[2]);
                            }
                        }
                    }
                }
            }
        }
        for(jr01=0;jr01<10;jr01++) {
            for(ic01=0;ic01<10;ic01++) {
                sum = 0;
                for(ic02=0;ic02<8;ic02++) {
                    for(jr02=0;jr02<8;jr02++) {
sum += (int)
floor(tempspread[jr01*8+jr02][ic01*8+ic02]+0.5);
                    }
                }
                sum = sum/64;
                fprintf(pixelmap,"%5ld ",sum);
            }
            fprintf(pixelmap,"\n");
        }
fprintf(pixelmap,"\n%5ld %5ld maxsum and minsum\n",maxsum,minsum);
}
void main(int argc,char *argv[]) {
int    im,ifc;
float scale; /* read in file names */
    if(argc<4) {
        printf("This program requires a .TDF, .PMP, and .NSE");
```

```
        printf(" file in the command line\n");
        getch();
        exit(0);
    }
    strcpy(name1,argv[1]);
    if((tdffile = fopen(name1,"r")) == NULL) {
printf("Could not open test target description file (%s)\n",name1);
        getch();
        exit(0);
    }
    strcpy(name2,argv[2]);
    if((pixelmap = fopen(name2,"w")) == NULL) {
        printf("Could not open pixel map %s\n",name2);
        getch();
        exit(0);
    }
    strcpy(name3,argv[3]);
    if((noisemodel = fopen(name3,"r")) == NULL) {
        printf("Could not open noise model %s\n",name3);
        getch();
        exit(0);
    }

    fscanf(tdffile,"filename %s",name3);
    fscanf(noisemodel,"filename %s",name3);
    fscanf(noisemodel," %f %f %f",&coefs[0],&coefs[1],&coefs[2]);
    fprintf(pixelmap,"filename %s\n",name2);
    tInit3(); /* set up graphics */
    Init3D(6);
    SetWorldCoordinates(-10.0,-10.0,10.0,10.0);
    LoadTargets();
    frame_cnt = 0;
    old_f_cnt = 1;
    if(no_frames > 0) {
        scale = (1.0/(float) no_frames);
        WorldScale3(scale,scale,scale);
        for (im = 0; im < no_images; im++) {
            for(ifc = 0; ifc < fcnt[im]; ifc++) {
                frame_cnt++;
                scale = ((float)frame_cnt/(float)old_f_cnt);
                WorldScale3(scale,scale,scale);
                WorldRotate3(rot[im],axn[im]);
                if(frame_cnt <= no_frames) {
                    _clearscreen(_GVIEWPORT);
                    DrawAndSaveAimPoint();
                    _clearscreen(_GVIEWPORT);
                    DrawTarget(im);
                    add_noise();
                    SaveMap(frame_cnt);
                }
                old_f_cnt = frame_cnt;
            }
```

```
        }
    }
    else printf("error - number of frames = 0\n");
    Close3DGraphics();
    fclose(tdffile);
    fclose(pixelmap);
}
```

```
/*
        filename: P_AIMPTS
        This program accepts an aimpoint list and
        computes the errors for plotting.
*/

# include <math.h>
# include <graph.h>
# include "worlddr.h"
# include <stdio.h>
# include "plot3d.h"
# include "hpplot.h"
# include "stdlib.h"

FILE    *aimpts;                /* asat aimpoints          */
FILE    *fzypts;                /* fuzzy aimpoints          */
FILE    *manpts;                /* manual aimpoints          */
FILE    *allpts;                /* all aimpoints          */
long    map[10][10];
char    name0[20];               /* test file name          */
char    name1[20];               /* asat aimpoint file name */
char    name2[20];               /* fuzzy aimpoint file name */
char    name3[20];               /* manual aimpoint file name */
char    name4[20];               /* all aimpoint file name */
int     num_maps;
int     map_no;
float   erout[10][10];
void ComputeError(int maps,int im)
{ /* compute ASAT aimpoint errors */
int    jpc,ipr;
float  jbc,ibr,jcc,icr,erb,erc;
double dbc,dbr,dcc,dcr,dpc,dpr,dnm;
double dtemp1,dtemp2,dtemp3;

    dnm = (float) (maps - im);

fscanf(aimpts," %d %d %f %f %f %f",&jpc,&ipr,&jbc,&ibr,&jcc,&icr);
    dbc = (double) jbc;
    dcc = (double) jcc;
    dpc = (double) jpc;
    dbr = (double) ibr;
    dcr = (double) icr;
    dpr = (double) ipr;
    dtemp1 = pow((dpr-dbr)*dnm,2.0);
    dtemp2 = pow((dpc-dbc)*dnm,2.0);
    dtemp3 = dtemp1 + dtemp2;
    erb = (float) pow(dtemp3,0.5);
    dtemp1 = pow((dpr-dcr)*dnm,2.0);
    dtemp2 = pow((dpc-dcc)*dnm,2.0);
    dtemp3 = dtemp1 + dtemp2;
    erc = (float) pow(dtemp3,0.5);
    erout[im][0] = erb;
```

```
    erout[im][1] = erc;

}
void main(int argc,char *argv[]){
int     im; /* read in file name */
char   s[80];
    if(argc<3) {
        printf("This program requires a test");
        printf(" file name in the command line.\n");
        getch();
        exit(0);
    }
    strcpy(name1,argv[1]);
    if((aimpts = fopen(name1,"r")) == NULL) {
        printf("Could not open asat aimpoint file (%s)\n",name1);
        getch();
        exit(0);
    }
    strcpy(name2,argv[2]);
    if((allpts = fopen(name2,"w")) == NULL) {
        printf("Could not open all aimpoint file (%s)\n",name4);
        getch();
        exit(0);
    }
    fscanf(aimpts,"filename %s",s);
    fscanf(aimpts," %d number of maps",&num_maps);
    for(im=0;im<num_maps;im++) {
       ComputeError(num_maps,im);
    } /* compute ASAT aimpoint errors */
    for(im=num_maps-1;im>=0;im--) {
fprintf(allpts,"    %f    %f\n\n",erout[im][0],erout[im][1]);
    }
}
```

```
/*
        filename: SHOW
        This program accepts a target description
        file and outputs image.
*/
#define   MAX_IMAGES   100
#define   MAX_SURFACES 10
# include <math.h>
# include <graph.h>
# include "worlddr.h"
# include <stdio.h>
# include "plot3d.h"
# include "hpplot.h"

int       err,i,j;
int       color;
struct    WorldRect wr;

float     x,y,z;
float     apx,apy,apz;
float       rot[MAX_IMAGES],xrot,yrot,zrot;
int       axn[MAX_IMAGES],colorno[MAX_SURFACES][MAX_IMAGES];
int       fcnt[MAX_IMAGES];
float     p[4][3][MAX_SURFACES];
int       no_images,no_surfaces,no_frames;
int       old_f_cnt,frame_cnt;
FILE      *tdffile;
char      name0[20];
char      name1[20];

void add_noise(void)
{
    return;
}
void DrawTarget(int no_surf,int image_no,int frame_no) {
struct  point3D pv[5];
int       isf,ipt;
/* draw image of target */
    for(isf=0;isf<no_surf;isf++) {
        for(ipt=0;ipt<4;ipt++) {
            pv[ipt].x = p[ipt][0][isf];
            pv[ipt].y = p[ipt][1][isf];
            pv[ipt].z = p[ipt][2][isf];
        }
        pv[4].x = p[0][0][isf];
        pv[4].y = p[0][1][isf];
        pv[4].z = p[0][2][isf];

        SelectColor(colorno[isf][image_no]);
        PolyFill3D(pv,1,colorno[isf][image_no],5);
    }
}
```

```
void LoadTargets(void)
{ /* read target description file */
int   im,is,ip;
int   image,surface;
char  title[80];

    fscanf(tdffile,"filename %s\n",title);

    fscanf(tdffile,"%d number of surfaces\n",&no_surfaces);

    for(is=0;is<no_surfaces;is++) {
        for(ip=0;ip<4;ip++) {
            fscanf(tdffile,"%f %f %f\n",
            &p[ip][0][is],&p[ip][1][is],&p[ip][2][is]);
        }
    }

    fscanf(tdffile,"%f %f %f aimpoint\n",&apx,&apy,&apz);

    fscanf(tdffile," %f %f %f xrot yrot zrot",&xrot,&yrot,&zrot);
    WorldRotate3(xrot,0);
    WorldRotate3(yrot,1);
    WorldRotate3(zrot,2);

    fscanf(tdffile,"%d number of frames\n",&no_frames);

    fscanf(tdffile,"%d number of images\n",&no_images);

    for(im=0;im<no_images;im++) {
fscanf(tdffile,"\n%d %f %d %d image rotation axes fcount\n",
                    &image,&rot[im],&axn[im],&fcnt[im]);
        for(is=0;is<no_surfaces;is++) {
fscanf(tdffile,"\n%d    %d    surface    color\n",    &surface,
&colorno[is][im]);
        }
    }
}
void main(int argc,char *argv[]) {
int    im,ifc;
float scale; /* read in file names */
    if(argc<2) {
        printf("This program requires a test");
        printf(" file name in the command line\n");
        getch();
        exit(0);
    }
    strcpy(name1,argv[1]);
    if((tdffile = fopen(name1,"r")) == NULL) {
printf("Could not open target description file (%s)\n",name1);
        getch();
        exit(0);
    }
```

```
    tInit3();
    Init3D(6);
    SetWorldCoordinates(-10.0,-10.0,10.0,10.0);
    LoadTargets();
    frame_cnt = 0;
    old_f_cnt = 1;
    if(no_frames > 0) {
        scale = (1.0/(float) no_frames);
        WorldScale3(scale,scale,scale);
        for (im = 0; im < no_images; im++) {
            for(ifc = 0; ifc < fcnt[im]; ifc++) {
                _clearscreen(_GVIEWPORT);
                frame_cnt++; /* display image sequences */
                scale = ((float)frame_cnt/(float)old_f_cnt);
                WorldScale3(scale,scale,scale);
                WorldRotate3(rot[im],axn[im]);
                if(frame_cnt <= no_frames) {
                    DrawTarget(no_surfaces,im,frame_cnt);
                }
                old_f_cnt = frame_cnt;
                getch();
            }
        }
    }
    else printf("error - number of frames = 0\n");
    Close3DGraphics();
    fclose(tdffile);
}
```

```
/*
        filename: SHOW_3
        This program accepts a test target description
        file and outputs three views.
*/
#define   MAX_IMAGES   100
#define   MAX_SURFACES 10
# include <math.h>
# include <graph.h>
# include "worlddr.h"
# include <stdio.h>
# include "plot3d.h"
# include "hpplot.h"

int       err,i,j;
int       color;
struct    WorldRect wr;

float    x,y,z;
float    apx,apy,apz;
float      rot[MAX_IMAGES];
int      axn[MAX_IMAGES],colorno[MAX_SURFACES][MAX_IMAGES];
float    p[4][3][MAX_SURFACES];
int      no_surfaces;
FILE     *tdffile;
char     name0[20];
char     name1[20];
/* draw image of target */
void DrawTarget(int no_surf,int image_no) {
struct  point3D pv[5];
int      isf,ipt;

    for(isf=0;isf<no_surf;isf++) {
        for(ipt=0;ipt<4;ipt++) {
            pv[ipt].x = p[ipt][0][isf];
            pv[ipt].y = p[ipt][1][isf];
            pv[ipt].z = p[ipt][2][isf];
        }
        pv[4].x = p[0][0][isf];
        pv[4].y = p[0][1][isf];
        pv[4].z = p[0][2][isf];

        SelectColor(isf+1);
        PolyFill3D(pv,1,isf+1,5);
    }
}
void LoadTargets(void)
{ /* read target description file */
int   is,ip;
char  title[80];

    fscanf(tdffile,"filename %s\n",title);
```

```
    fscanf(tdffile,"%d number of surfaces\n",&no_surfaces);

    for(is=0;is<no_surfaces;is++) {
        for(ip=0;ip<4;ip++) {
            fscanf(tdffile,"%f %f %f\n",
            &p[ip][0][is],&p[ip][1][is],&p[ip][2][is]);
        }
    }
}
void main(int argc,char *argv[]) {
int    im,ifc; /* read file names */
float scale;
    if(argc<2) {
        printf("This program requires a target");
        printf(" description file name in the command line\n");
        getch();
        exit(0);
    }
    strcpy(name1,argv[1]);
    if((tdffile = fopen(name1,"r")) == NULL) {
printf("Could not open target description file (%s)\n",name1);
        getch();
        exit(0);
    }
    LoadTargets(); /* setup three D views */
    rot[0] = -90.0;
    axn[0] = 1;
    rot[1] = 90.0;
    axn[1] = 1;
    rot[2] = 90.0;
    axn[2] = 0;
    tInit3();
    Init3D(6);
    SetWorldCoordinates(-10.0,-10.0,10.0,10.0);
    for (im = 0; im < 3; im++) { /* draw three views of target */
        _clearscreen(_GVIEWPORT);
        WorldRotate3(rot[im],axn[im]);
        DrawTarget(no_surfaces,im);
        getch();
    }
    Close3DGraphics();
    fclose(tdffile);
}
```